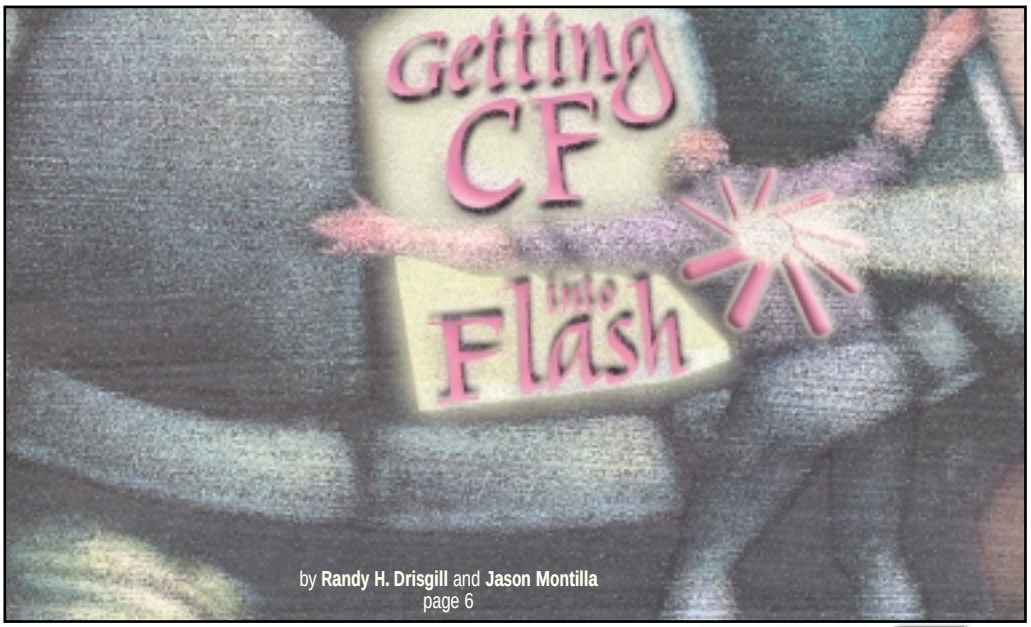
U.S. $8.99 (Canada $9.99)

ColdFusion Developer's Journal

ColdFusionJournal.com

August 2001 Volume: 3 Issue: 8



by Randy H. Drisgill and Jason Montilla
page 6

SYS-CON MEDIA

ColdFusion Developer's Journal

subscriptions:
For subscription requests please call
1 800 513-7111 or go to: **www.sys-con.com**
domestic $89.99/yr. (12 issues)
canada/mexico $99/yr
overseas $129/yr
back issues $12 U.S. $15 all other

editorial offices: SYS-CON MEDIA, INC.
135 Chestnut Ridge Rd., Montvale, NJ 07645
Telephone: 201 802-3000 **Fax:** 201 782-9600
COLDFUSION DEVELOPER'S JOURNAL (ISSN #1523-9101)
is published monthly (12 times a year)
for $89.99 by **SYS-CON Publications, Inc.,**
135 Chestnut Ridge Rd.,Montvale, NJ 07645

postmaster: send address changes to:
COLDFUSION DEVELOPER'S JOURNAL
SYS-CON MEDIA
135 Chestnut Ridge Rd.,Montvale, NJ 07645

worldwide distribution:
by **Curtis Circulation Company**
New Milford, NJ 07646-3048

distribution in USA:
by **International Periodical Distributors**
674 Via De La Valle, Suite 204, Solana Beach, CA 92075
Phone: 619 481-5928

CF Advisor and CF Fast Track

BY **ROBERT DIAMOND**

We've got some exciting news to share with you before you get into this information-packed issue of ***ColdFusion Developer's Journal***.

CF Advisor

SYS-CON Media, the leading publisher of *i*-technology magazines and the parent company of ***CFDJ***, recently aquired ***CF Advisor*** and has added it to its online stable of Web sites. No changes will be made to the Web site, which will continue in its role as an independent ColdFusion developer's resource. Complete with news, tips, feature articles, a job bank, training resources, a CF hosting guide, and discussion areas, ***CF Advisor*** is one of the leaders in the online world of ColdFusion development.

Rod Bicknell will continue in his role as editor-in-chief of the online publication. He's done a great job in the past, and we're all looking forward to working with him in the future as ***CF Advisor*** continues to grow alongside ***CFDJ***.

Also, look for all of the great ***CF Advisor*** content coming soon on CD via JDJStore.com. A perfect companion to the ***CFDJ*** Complete Works Archives CD as a fully functional ColdFusion resource right at your fingertips.

ColdFusion Fast Track at JDJEdge 2001

Also happening behind the scenes for those not yet aware, we've announced the final technical program for the ColdFusion Fast Track, complete with cutting-edge technical sessions, to be held in New York City, September 23–26, 2001. Get all the information you need at www.sys-con.com/coldfusionedge. Updated daily with the latest conference news and announcements, including details of the ColdFusion Technology Pavilion on the show's exhibit floor, it's worth checking out. If you can make it in September, you won't be disappointed. Hope to see you there!

On to This Month's Issue...

Coauthors Randy Drisgill and Jason Montilla, along with Dennis Baldwin – a two-punch combo – have contributed remarkable pieces on using ColdFusion with Flash. The first piece, "ColdFusion-Driven Flash Content" (Part 1 of 2), covers the advantages of using Flash, then dives into the basics of getting ColdFusion data into Flash. Part 2, in an upcoming issue, will teach you how to send data in and out of CF and Flash using a database. For those eager to jump right in, "ColdFusion Meets Flash" goes into the basics of loading in variables from ColdFusion, querying database outputs, and moving data out to create a dynamic navigation menu. Both pieces provide solid foundations for those new to Flash. If you haven't used Flash before, or integrated it with ColdFusion, you'll be surprised by two things: (1) how easy it is, and (2) how much better you can make your site look by using it.

The balance of the issue contains a wealth of knowledge as well. "Creating Intelligent 404 Pages," by Mark Carson, covers not only specifying custom 404 error pages, but also creating a page in CF to e-mail you automatically and log the error as it happens. Curtis Schlak covers the creation and use of Web wizards; Kelly Brown writes on how to send HTML e-mail messages complete with the ColdFusion code. As always, Bruce Van Horn's "Ask the Training Staff" answers your latest <cfquestions>.

All this plus the usual columns from Ben Forta and others....See you in September!

ABOUT THE AUTHOR

Robert Diamond is editor-in-chief of **ColdFusion Developer's Journal** *as well as* **SYS-CON**'s *newest magazine,* **Wireless Business & Technology**. *Named one of the "Top thirty magazine industry executives under the age of 30" in Folio magazine's November 2000 issue, Robert recently graduated from the School of Information Studies at Syracuse University with a BS in information management and technology.*

ROBERT@SYS-CON.COM

BY **RANDY H. DRISGILL** AND **JASON MONTILLA**

With the recent merger of Allaire and Macromedia – the combined concern now called Macromedia – developers should be excited about all the new ways ColdFusion development technologies can be used with Flash interface technologies.

We felt it was time to show how easy it is to get the two to communicate with each other. Since Macromedia came out with Flash version 5.0, the ActionScript programming language has finally reached an acceptable level of usability. ActionScript is an object-oriented language similar to JavaScript that allows developers to create dynamic and intelligent Flash applications.

For these articles we'll be using ColdFusion 4.5 with Flash 5.0, and we expect you to have a basic understanding of the ColdFusion programming language, basic Flash ActionScript development, and ColdFusion 4.5 server administration.

Reasons to Use Flash

Before we dive into the code, you may ask why you would even want to use Flash as a graphical interface for ColdFusion. Actually there are several good reasons:

- Flash is an excellent medium to display text, graphics, and animation in a compact, scalable format. It's a vector animation tool, which means that native objects are calculated from mathematical equations rather than bitmaps. You can also import bitmap images into Flash, but it's best to stick with objects in vector format for file size reasons.
- Flash has some inherent benefits from a security standpoint. Since Flash uses .swf files that are compiled from a source .fla file, Web browsers can't simply right-click and view source to see everything you're doing. For example, a login script can call other Flash files or a ColdFusion template to validate the user ID and password, but the end user will never see the actual name of the processing file or any of the URL parameters.
- On the same note, Flash/ColdFusion sites can be developed so the end user never sees a .cfm file extension. The technology can be masked behind a simple .html page with an embedded flash .swf file.
- Macromedia is beginning to position Flash not only as a cross-browser display tool, but as a cross-device/cross-platform display tool. Flash already runs on PocketPCs and PlayStation 2s, and we can assume that only more devices like these will be announced in the future.
- Flash's scalable cross-browser format allows you to focus more on coding the application than deciding how to display it on noncompliant browsers and low-resolution displays. Plus, Macromedia has made sure that a majority of today's browsers are already installed with the Flash plug-in.
- Finally, Flash maintains a persistent client-side connection to the variables. In other words, from page to page, variables don't need to be passed. This is beneficial because the database needs to be queried only one time to gather all the information that Flash will use during that session.

We hope you're convinced now that Flash is a viable solution for displaying ColdFusion content, so let's get started.

Plan of Action

Part 1 focuses on the basics of getting ColdFusion data into Flash, and Part 2 will offer a simple application that will demonstrate the process of sending data in and out of ColdFusion using a database to store user input. We begin by simply reading a text file into a Flash variable and displaying the contents in a Flash dynamic text box.

Next, we substitute a ColdFusion template for the text file, allowing the content to update dynamically. It should be noted that there are other ways to bring variables into Flash utilizing technologies such as XML and WDDX, but we won't be covering that here.

The concepts in this article have been developed for the Flash ActionScript novice to appeal to the widest audience, and to allow you to get up and running quickly.

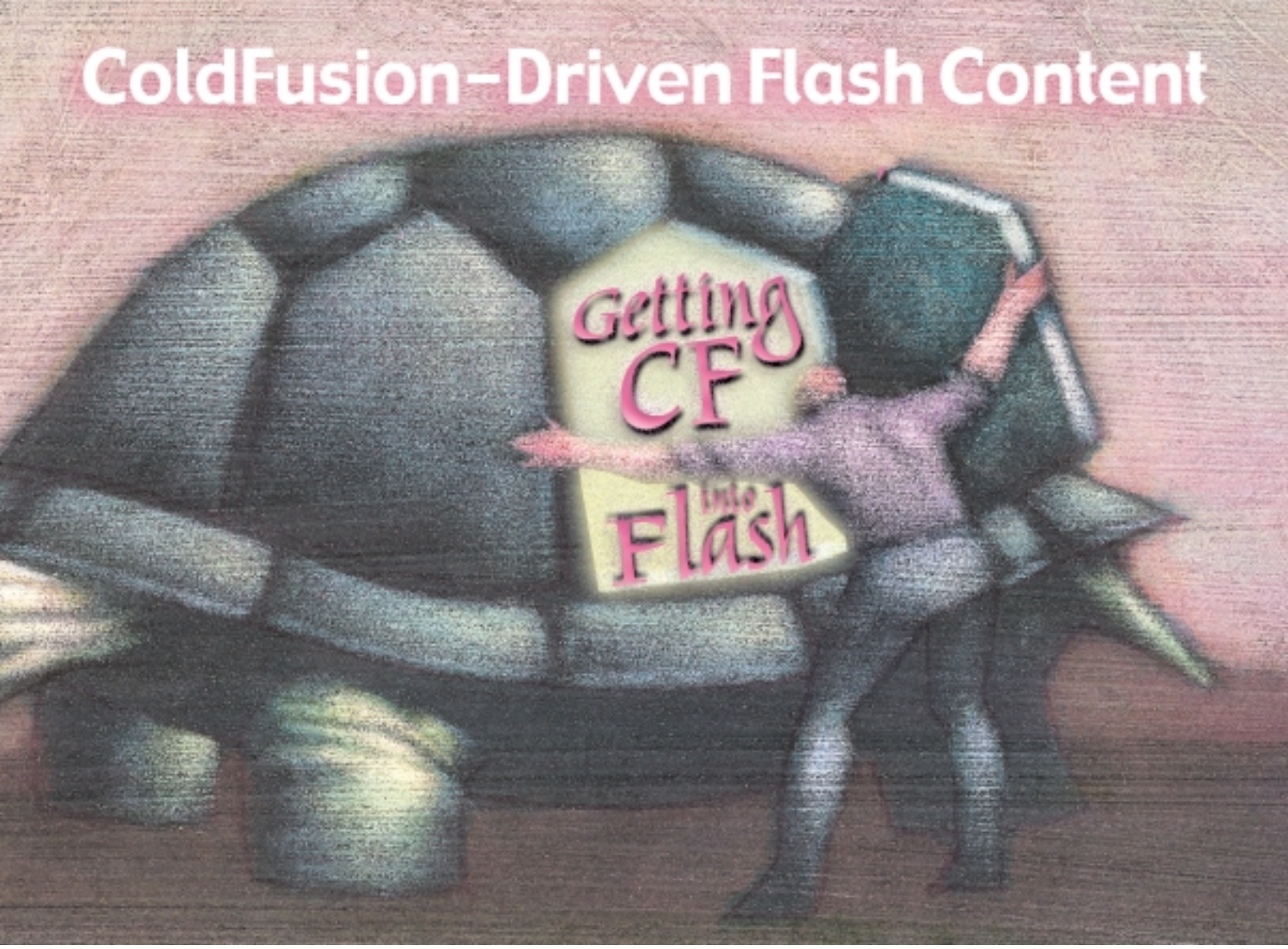
Getting Started

First, open up Flash 5.0 and create a new movie file. The default settings for a new Flash file (.fla) create only one layer called *Layer 1*. We'll create an extra layer to keep our development environment organized.

Directly under the layers there are three buttons: a white box with a plus (+), a blue box with a tilde (~), and a garbage can. Click on the + to create a new layer called *Layer 2*. If you double-click the layers, you can rename them; let's call ours *code* and *info* (see Figure 1).

We'll use *code* for all our ActionScript scripts and *info* for the visual objects. Next, we create four keyframes in the code layer, and one keyframe on the fourth frame of the info layer. You can easily create keyframes by right-clicking on a frame and selecting "insert keyframe."

ColdFusion-Driven Flash Content



Also, double-check the second and third frames of the info layer and make sure they're *not* keyframes by selecting them and right-clicking and choosing "Clear Keyframe." This is important because the objects won't display properly if all the frames of the info layer are keyframes.

Placing the Visual Elements

Now that we have the layers and keyframes set up, we can begin placing the visual elements on the stage. First we want to create a large text box in frame 4 of the info layer. We'll need to edit some of the properties of the text box using the "Text Options Panel" (see Figure 2).

In this panel we need to select "Dynamic Text," "Multiline," "Selectable," and "Word Wrap." This allows the contents of the text box to be driven from a variable, be more than one line, be clickable, and wrap text from one line to the next.

We also need to set the name of the variable that will drive the content. Make sure you have the text box selected and enter the name "dynamictext" in the Variable field of the "Text Options Panel."

If we were to set the value of the variable dynamictext, it would replace any text already in the text box. Instead of doing this, we'll load the value of the variable from a text file.

The ActionScript Component

Let's move on to the ActionScript that will read from the external text file. Double-click on frame 1 of the code layer; this will open the internal ActionScript editor in Normal Mode, the default editing mode. We recommend using Expert Mode (Ctrl-E) to edit your code, since this gives you considerably more flexibility in editing. Let's add the line of code that reads the variables.txt file (see Figure 3).

We'll place our text file in the same directory as our .swf file, so we don't need to enter any directory information to find the file. If the text file were located in a different directory, we'd be able to load it using either absolute or relative filenames.

The second parameter, "", can be used later to target specific areas of your Flash movie; we'll be using "" to signify the current location.

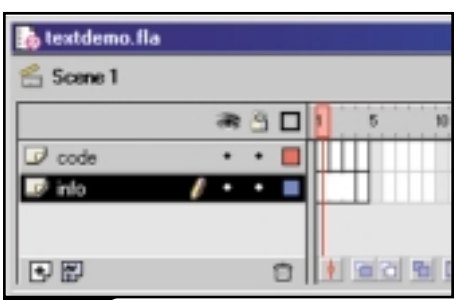


FIGURE 1: Layer 2 movie file

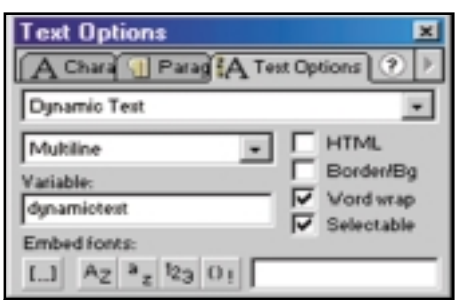


FIGURE 2: Text Options Panel

We can check the syntax of our code by quickly switching the mode to Normal Mode (Ctrl-N). Any errors in the Flash code will be displayed to us immediately. If nothing is shown, Flash will reorganize the code and we can assume the syntax is correct (at least as far as Flash syntax goes). Then we switch back to "Expert Mode" (Ctrl-E).

Let's skip to frame 3 of the code layer and enter some more ActionScript.

```
if (datacomplete == "Yes") {
  gotoandplay (_currentframe+1)
} else {
  gotoandplay (_currentframe-1)
}
```

This code is necessary because Flash will process very quickly, and most likely try to continue processing even before the text file is completely loaded (especially if there's a slow connection or a large text file).

Using the datacomplete Variable

We utilize the variable datacomplete, which will be set to "Yes" in the text file. This will be the final variable sent from the text file, so logically if we wait until this variable is read, we'll know everything is loaded from the text file. Then we can advance from frame 3 to frame 4.

If the variable doesn't equal "Yes" (which will be the case if it hasn't been loaded), our code will tell Flash to go to the previous frame. You'll notice that we purposely didn't put anything in frame 2 of the code layer. This causes Flash to continually loop between frame 2 and frame 3 until the datacomplete variable is completely read. If any ActionScript were located in frame 2, it would be executed each time the frame was loaded.

Frame 4 of the code layer will simply have a "stop ();" command in the ActionScript to ensure that the movie doesn't loop back to the beginning. This is really all we need from the Flash side of things to accomplish the text file load. However, since frames 1–3 of the info layer are blank, and won't show anything while the text file is loading, we should add a visual so users know something is happening.

Simply select frame 1 of the info layer and add some text to the stage that says, "Loading Data." That's it! Save the file as textdemo.fla.

Creating the Text File

Now we have to create the text file, variables.txt., in the same directory as the .fla file. Note that Flash is very specific about the formatting in the text file. Errors won't be displayed in a meaningful error statement, so care needs to be taken when creating the file.

One way to think of the text file is as a URL string of variables and values; this is how Flash expects the variables. One rule of thumb is to create variables with an ampersand in front of the name like this: "&dynamictext=The quick brown fox jumped over the lazy dog." Flash doesn't allow spaces between the parts of the statement, though the variable can have spaces in it. You can't have spaces after the ampersand, variable name, or equal sign, or between the statements.

We'll then add our "&datacomplete=Yes," which will be our last variable. There can't be anything after the datacomplete variable is set. No carriage returns or spaces will be accepted after all the variables are defined.

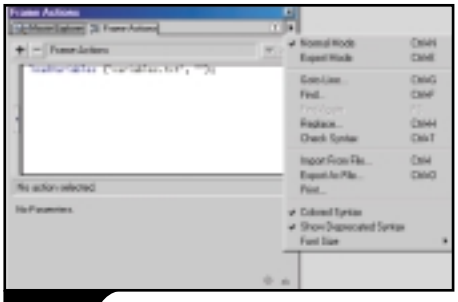


FIGURE 3: The ActionScript editor

Inspecting the File

Once again, Flash won't display anything if there are any problems, so after the file is created, you may want to view it in a browser and make sure there are no extra spaces or carriage returns. Variables are treated similar to HTML <pre> tags; one carriage return in the variable will make a new paragraph in the Flash text box. If you need more control than this, you can select "HTML" in the "Text Options Panel," and then make sure you enter HTML with
s in your text file.

Our text file looks like this:

```
&dynamictext=The quick brown fox
  jumped over the lazy dog.&datacom
  plete=Yes
```

We can now test the file in Flash, and the text should fill up the text box. To try the file from a Web browser, publish the Flash file to an .swf file and place it in the same directory as the text file.

That wasn't too hard, but don't be surprised if things don't work the first time you try them. Flash is very particular about how things should be, and is not very friendly when it comes to error statements. Stick with it, and make sure everything is precisely how it should be.

Hopefully everything worked for you and we can now focus on changing the text file to ColdFusion, and make some quick changes to the Flash movie to call the ColdFusion file instead of the text file.

To Call the ColdFusion File

First we'll change frame 1 of the code layer to load a .cfm file instead of the .txt file. Also, let's save the Flash file as "cftextdemo.fla," export a movie called "cftextdemo.swf," and place both of them in the same directory as before.

Your code should be as follows:

```
loadVariables ("cfvariables.cfm", "");
```

Create a simple ColdFusion template called "cfvariables.cfm," which will simply set a variable "thevar" to our text, "The quick brown fox jumped over the lazy dog." We'll also include one special tag called <cfsetting> that will be used to turn off any output other than that in a <cfoutput> tag and will suppress any debug information for just this one page.

This is important because the debug info and all the white spaces that ColdFusion creates will instantly break a Flash application. *Note:* We put the variable string inside <cfoutput> tags because of the <cfsetting> tag, which only shows output inside of the <cfoutput> tags.

Your code will be as follows:

```
<cfsetting enablecfoutputonly="Yes"
    showdebugoutput="No" catchexcep
    tionsbypattern="No">
<cfset thevar = "The quick brown fox
    jumped over the lazy dog">
<cfoutput>&dynamictext=#thevar#&data
    complete=Yes</cfoutput>
```

You're All Set

This is all we need to do. Now that the text file is being created dynamically, try it out in a Web browser. Once you have this simple example perfected, you should be able to do some really interesting things with queries and the like.

Part 2 will focus on creating an actual application that displays information from a database and takes user input from Flash to update the database.

About the Authors

Randy H. Drisgill is the CTO and cofounder of Vshift (a premier consulting partner). An Allaire Certified Professional, he's been using ColdFusion since 1998 and has extensive experience with data-driven Web applications, content management solutions, Web design, mobile applications, and data-driven Flash applications.

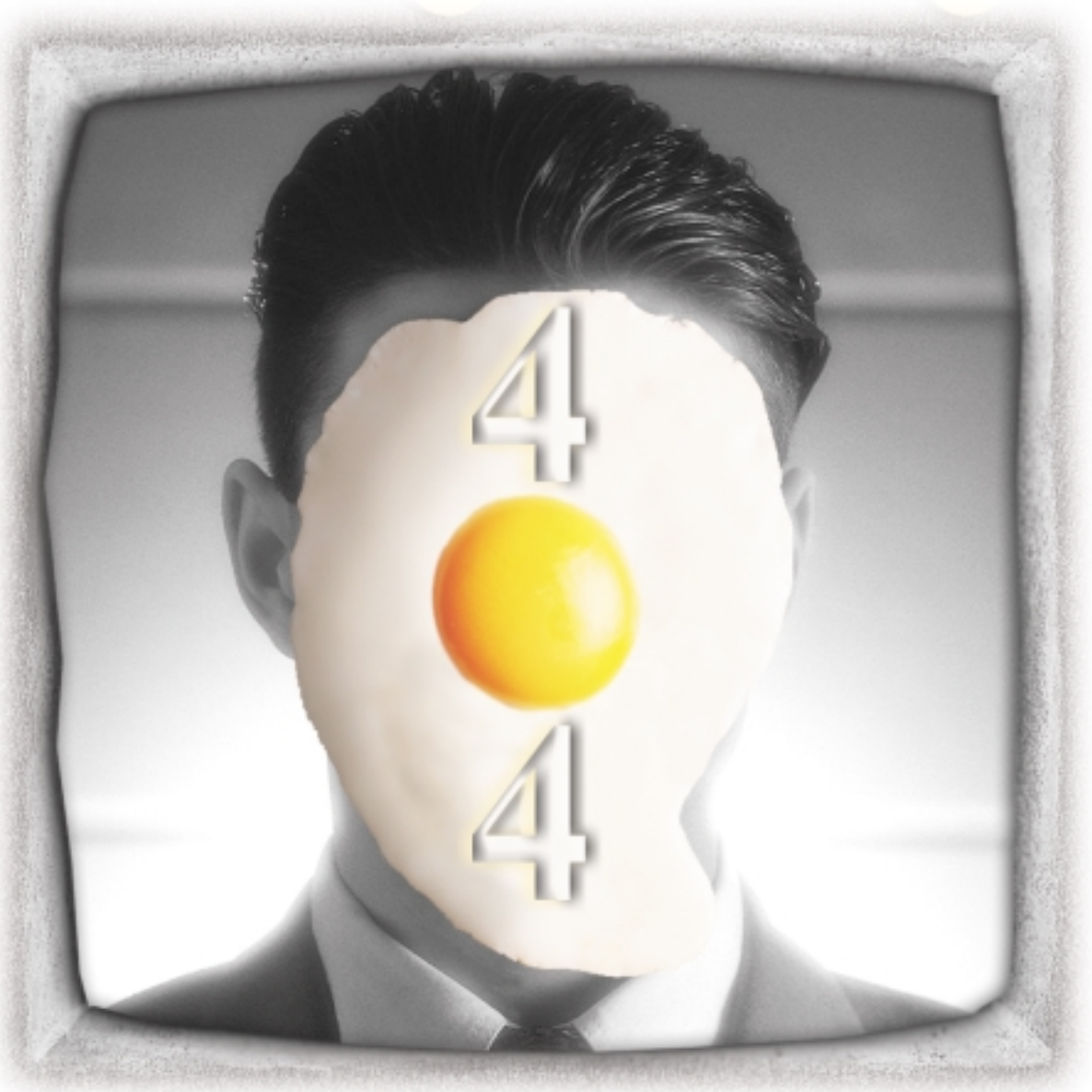
Jason Montilla, an Allaire Certified Professional, works for Vshift. He's been using ColdFusion for two years and Flash for four, and is currently building several Web and mobile data-driven Flash applications.

rdrisgill@vshift.com

jmontilla@vshift.com

Creating Intelligent 404 Pages

BY MARK CARSON



Doing useful things with a custom 404 page

In the age of the Internet, even obscure acronyms and codes have become commonplace. Take "404", for example. Even Internet newbies associate the number "404" with that annoying error page that is displayed when searching for a reliable doggie dentist. In fact, there are even numerous Web sites dedicated to the "art" of 404 pages (www.sendcoffee.com/minorsage/404error.html).

However, for developers and administrators, 404 pages on our own sites are not regarded as humor, but represent an aggravating inevitability that 404s *do* happen. Most Web site administrators are aware of the capability of substituting an alternate 404 page through their Web server, but most haven't realized the intelligence and utility that can be built into it with a few snippets of code.

This article is intended to give an overview of just a few of the cool and useful things you can do with a custom 404 page. My tool of choice in this article is ColdFusion and IIS; however, most of these techniques can also be applied with other popular Web technologies such as Perl, ASP, and JSP.

Missing Pages

It's as inevitable as death and Microsoft: files get deleted, directories get moved, and your visitors get 404 pages. The most common use for a custom 404 page is to simply convey a more user-friendly message to your visitors. By default, most Web servers are configured to deliver eloquent public relations material like "HTTP 404 – FILE NOT FOUND." It's ugly, it's uninformative, and it's a real turnoff for your visitors. At the very least, you should create a custom 404 page that has your company logo and some polished explanation as to why they didn't get the page they were looking for.

"It's as inevitable as death and Microsoft: files get deleted, directories get moved, and your visitors get 404 pages"

How to Specify a Custom 404 Page

Microsoft IIS 4.0

For the techniques in this article to function properly, you'll need to make two configuration changes to your Web server: one to IIS and one to ColdFusion.

From the Internet Service Manager Console of IIS, choose "Properties" for your Web site. Click the tab labeled "Custom Errors." Scroll down to the HTTP Error named "404", select it, and click on the "Edit Properties" button (see Figure 1). From the edit window, choose "URL" from the drop-down menu and specify the path to your custom 404 template in the contents field. Click "OK" and apply your new settings. You should be able to see the new settings at work by typing in a nonexistent page on your site.

ColdFusion 4.5

From the ColdFusion Administrator click on "Settings." In the field labeled "Missing Template Handler," specify the complete path to your custom 404 template. This should be the same file that you specified in the IIS settings (see Figure 2).

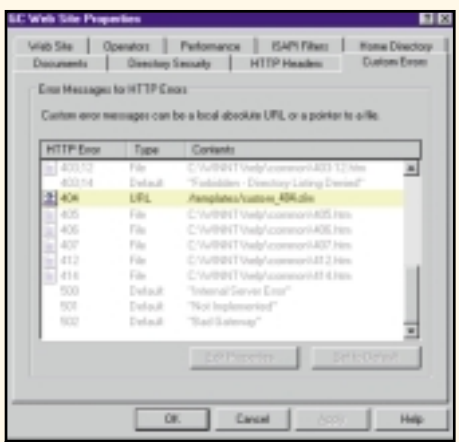


FIGURE 1: IIS configuration

FIGURE 2: ColdFusion 4.5 configuration

However, the best 404 page by far is the one that's avoided altogether. Add these easy functions to your custom 404 page to help in the war against Web site obsolescence.

Most Web site administrators have access to log analysis tools, such as WebTrends, that can report on information such as missing pages. However, in the course of the weekly rat race, many administrators and developers simply can't keep pace with all the changes that take place on their site. Here are a few ideas to keep administrators and developers informed:

1. Anytime the custom 404 page is encountered, you can have it send you an e-mail that specifies the requested file (see Listing 1). This gives you near real-time feedback to what your customers are experiencing. The only drawback to this is if you have a large site with a lot of traffic; the potential exists for you to get inundated with e-mail. But hey, that may provide just enough motivation to keep your site clean!
2. Log all missing pages to a data source. Once you have it in a database, you can create a variety of reports that can be accessed on demand, or better yet, delivered via e-mail each morning to every developer on the team.

Missing Images

Missing images on your site may not be as intrusive as missing pages, but they are still a major nuisance to your customers. With a little creative coding you can easily minimize the impact that missing images have on your site.

With the exact same custom 404 page, check to see if the missing file is a GIF or JPEG image. If so, specify a different content type and return a transparent 1x1 pixel using <CFCONTENT> (see Listing 2). It may not actually solve the problem, but you've eliminated the dreaded broken-image icon and, in some cases, the user won't even realize that anything is amiss. And, of course, you should still do the same logging techniques referenced above so that you can fix the root of the problem.

Template Conversion

Have you ever been forced to convert a site from one scripting language to another? ASP to ColdFusion? ColdFusion to JSP? It's a pain in the arse to begin with, but what about all those customers who have bookmarks created for specific pages on your site? There's nothing you can do, right? Wrong!

I recently converted a servlet-based site with ".shtml" file extensions to ColdFusion with ".cfm" file extensions, but the client demanded that I maintain all the old links that were being used by customers and partners. Once again, using the same custom 404 page, I created a function that checked all incoming file requests to see if they ended with ".shtml". If one did, I simply stripped off the file suffix and redirected it to its ".cfm" equivalent (see Listing 3).

Note: You'll need to make sure that the obsolete file extension is either processed by the ColdFusion dll or removed from the IIS Application Mappings completely. In this scenario, if I let ".shtml" files continue to be processed by the servlet engine, the redirect would never have the opportunity to execute.

Redirect Specific Files

Here's another related scenario: a user accesses a bookmarked page from an old version of your site, www.mysite.com/products/HERB.shtml. However, since the site is now built on ColdFusion and the product pages are generated dynamically, the bookmarked page would have three strikes against it:

1. The file extension ".shtml" is not supported.
2. The file www.mysite.com/products/HERB.shtml no longer exists.
3. www.mysite.com/products/HERB.cfm doesn't exist.

Here's where custom 404 pages get really powerful!

You could create a simple database table that holds the path of every key page on the old site along with its new destination. When the custom 404 page is called, do a query based on the requested page to see if there's a redirect entry for it (see Listing 4). If so, simply tell ColdFusion to do a <CFLOCATION> to the new destination. Therefore, without the customer ever knowing, the Web server has taken a request for www.mysite.com/products/HERB.shtml, determined that it's missing, performed a query to see if special handling was required, and redirected it to its new destination, www.mysite.com/products/handler.cfm?prod_id=425, all within a few hundred milliseconds!

Virtual Directories

Built on the same principal described above, another useful function for the custom 404 page is to create virtual directories. Biz Dev frequently requests special paths for various partnerships and promotions (e.g., www.mysite.com/dogs). Reluctant to clutter up the root directory with hundreds of directories that are not core to the site, I realized that by using the function referenced above I could create an unlimited number of "virtual" directories that redirect to a different destination. For instance, www.mysite.com/dogs redirects to www.mysite.com/promotions/december/dogs.cfm without the customer even knowing it. Best of all, the physical directory "/dogs" doesn't even exist!

About The Code Listings

The code in this article was written using ColdFusion 4.5.1 running on Microsoft NT 4.0 SP 5 with Microsoft Internet Information Server 4.0. For the complete code and instructions used for the listings, send an e-mail to cf_code@recursivemedia.com.

About the Author

Mark Carson is the chief Web architect for a prominent e-tailer and a certified ColdFusion developer.

mark@recursivemedia.com

Listing 1

```
<!--- Send off an e-mail to the admin --->
<cfmail to="webmaster@mysite.com"
from="webmaster@mysite.com"
subject="Missing File Encountered">
Date: #dateformat(now())#
Time: #timeformat(now())#
File: #CGI.PATH_INFO#
Referring Page: #CGI.HTTP_REFERER#
</cfmail>

<!--- Insert the missing page into a datasource --->
<cfquery name="insert_404"
datasource="my_db"
dbtype="ODBC">
INSERT INTO missing_files(date_added, file, referer)
VALUES(#now()#, '#CGI.PATH_INFO#', '#CGI.HTTP_REFERER#')
</cfquery>

<!--- Display a polished message --->
<html>
<head>
<title>Page Not Found</title>
<!--- Optional: Automatically redirect the customer to your
home page --->
<META HTTP-EQUIV='refresh' CONTENT='5; URL=/index.cfm'>
</head>
<body>
<br><br>
<table border="0" cellpadding="0" cellspacing="0"
width="502">
<tr><td align="center">
<img src="/images/logo.gif" width="200" height="50"><br>
<b><font face="Verdana" size="2">Sorry, but the page you
requested was not found.<br>
You will be automatically redirected to the <a
href="/index.cfm">home page</a> in 5 seconds.
</font></b>
</td></tr>
</table>
<br><br>
</body>
</html>
```

Listing 2

```
<cfif (#left(CGI.QUERY_STRING,3)# EQ 404) AND
(#right(CGI.QUERY_STRING, 4)# IS '.gif')>
   <cfcontent type="image/gif"
file="d:\inetpub\wwwroot\images\trans_pixel.gif" delete-
file="No">
   <cfabort>
<cfelseif (#left(CGI.QUERY_STRING,3)# EQ 404) AND
(#right(CGI.QUERY_STRING, 4)# IS '.jpg')>
   <cfcontent type="image/gif"
file="d:\inetpub\wwwroot\images\trans_pixel.gif" delete-
file="No">
   <cfabort>
</cfif>
```

Listing 3

```
<cfif #right(CGI.PATH_INFO, 6)# IS '.shtml'>
   <cfif #CGI.QUERY_STRING# IS ''>
      <cflocation URL="#replace(CGI.PATH_INFO, '.shtml',
'.cfm')#">
   <cfelse>
      <cflocation URL="#replace(CGI.PATH_INFO, '.shtml',
'.cfm')#?#CGI.QUERY_STRING#">
   </cfif>
</cfif>
```

Listing 4

```
<cfquery name="checkRedirect"
datasource="my_db"
dbtype="ODBC">
SELECT target_page
FROM redirect
WHERE current_page = '#CGI.PATH_INFO#'
</cfquery>

<cfif #checkRedirect.recordcount# GT 0>
   <cfif #CGI.QUERY_STRING# IS ''>
      <cflocation URL="#checkRedirect.target_page#">
   <cfelseif #findnocase("?", checkRedirect.target_page)#
GT 0>
      <cflocation
URL="#checkRedirect.target_page#&#CGI.QUERY_STRING#">
   <cfelse>
      <cflocation
URL="#checkRedirect.target_page#?#CGI.QUERY_STRING#">
   </cfif>
</cfif>
```

CODE LISTING

The code listing for this article is also located at

www.sys-con.com/coldfusion/sourcec.cfm

CF_Buster by CentraSoft

BY KELLY BROWN

The CF_Buster is a testing tool designed to help you pass the ColdFusion certification exam from Allaire. It provides simulated exams that are similar to the real computerized ones.

You can use CF_Buster to evaluate your ColdFusion skills and identify any areas that you need to improve.

The Details

CF_Buster has over 500 questions broken down into 18 topics. You have the option of taking one of eight predefined tests that cover all the topic s in the real test. Each predefined test consists of 61 questions with a time limit of one hour, as with the real test.

You can also take a random test for which you can define the number of questions and time limit.

Taking a predefined test is the best way to get started. It will give you an overview of your current skill level and identify any weak areas. After taking a practice test, you can brush up on any areas you had difficulty with and use a topic test to evaluate your progress.

The test is formatted similarly to the real test, as shown in Figure 1. You are given a screen with a question and several possible answers to choose from. Many questions have only one answer, but some questions require you to select more than one correct answer. There are typically five possible answers, but there can be any number of answers including true or false.

Each screen enables you to go to the next question or return to the previous one. In addition, you can mark questions you're not sure about and review them at the end of the test.

Since the test is timed, it's a good idea to do this. Unlike the real test, you can pause the timer and there's a help function that will show you the correct answer for the question you're currently on.

Once you complete the test, you receive a score. You need a score of 60% to pass the test. The score results show you a breakdown of the number of questions you got right and wrong in each area. You can also review all the questions, the correct answers, and your answers to see what you got wrong and why.

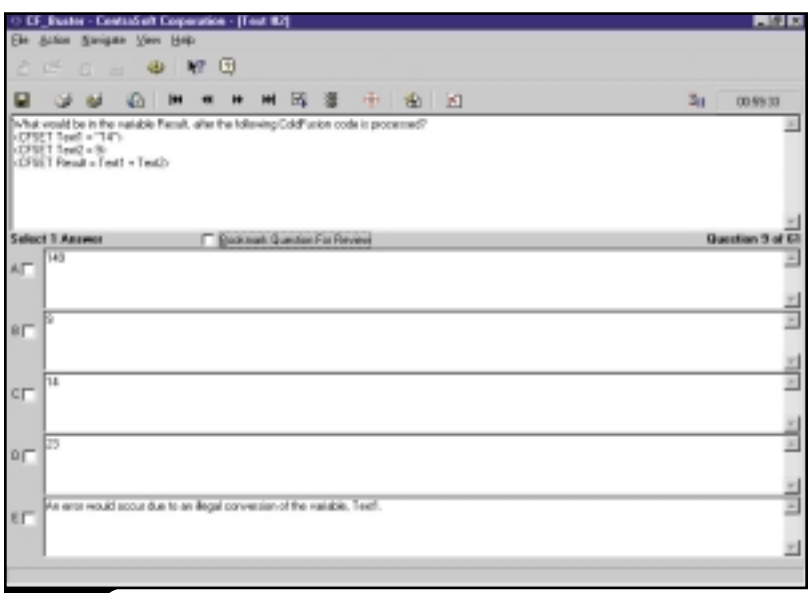


FIGURE 1: CF_Buster

VITALS

CF_Buster:
CentraSoft

Web Address:
www.centrasoft.com

Test Environment:
Windows

Pricing:
$39.95

Conclusion

CF_Buster is a good evaluation tool and will help you prepare for the test. It provides many questions from all the areas of the ColdFusion certification test. If you've never taken a computerized certification test before, just becoming familiar with the form and types of questions is valuable.

This product alone is not enough to prepare you for the test, however. There's no substitute for hands-on experience, and a study guide like *Certified ColdFusion Developer Study Guide* by Ben Forta will help you fill any gaps in your experience.

You can get more information on the Certified ColdFusion Developer Exam in the services area of Allaire's Web site under "training" at www.allaire.com/services/training/certification/index.cfm.

ABOUT THE AUTHOR

Kelly Brown is the CTO of About Web (www.aboutweb.com), an Internet solutions provider in the Washington, DC, area. He has a BS and MS in computer science and is a Microsoft-certified systems engineer.

KBROWN@ABOUTWEB.COM

Harnessing the **Power of** HTML E-Mails

Increase the impact of your e-mail advertising efforts

BY
KELLY BROWN

E-mail messages are one of the premier advertising techniques of the Internet era. However, plain text e-mails are no longer sufficient. Today, HTML messages make your advertisements stand out.

A well-designed HTML message or newsletter can give your organization a professional edge, and increase response rates from your advertisements.

Of course, HTML e-mail messages aren't some sort of silver bullet capable of solving all your advertising woes in one fell swoop. We've probably all seen ineffective HTML messages, the ones with the graphics that take too long to load, broken links, or even (gasp!) the dreaded blinking text.

With a little technical knowledge and a few design considerations, you can start sending your own professional high-impact e-mails using ColdFusion.

Sending HTML E-Mails the Easy Way

Sending an HTML e-mail message is easy in ColdFusion. The CFMAIL tag has a TYPE parameter that allows you to use HTML in your message. Simply set the type to "HTML" and place your HTML code in the message as shown below:

```
<cfmail to="audience@your
company.com"
from="kbrown@aboutweb.com" subject
="HTML Email" type="HTML">
…Place HTML Here …
</cfmail>
```

That was easy, but there's a catch. This technique sends an HTML e-mail to all the designated recipients. But what if they don't have an e-mail client that's capable of displaying HTML e-mail? For those poor individuals saddled with a text-only e-mail client, the message shows up with all the HTML tags displayed, making it difficult to read and irritating.

What do you do about these people? One solution is to flag them as either HTML or text recipients. Then you can send an HTML version of your message to some people and a text version to others. When people sign up for a newsletter or report, let them decide if they want to receive HTML e-mail or not.

Of course, you may not have a list in which people can select the type of e-mail they want, or a person may change his or her e-mail client. The ideal solution would be to send an e-mail that displays HTML for those recipients that support it and text for those who do not. This solution is possible, but it's more complex than the previous example.

```
Date: Sun, 03 Jun 2001 20:58:20 -0400
From: kelly@aboutweb.com
Subject: Simple HTML Test
To: audience@yourcompany.com
Content-type: text/html

<BODY>
This is a html email.<BR>
You can use html tags like <B>Bold</B>.
</BODY>
```

FIGURE 1: A sample HTML e-mail message

Understanding E-Mail

To understand how to send e-mail messages with both an HTML and a text version, you need to understand how e-mail works. E-mail messages, like Web pages, consist of a header and a body. The header is separated from the body of the message by a blank line. The header contains information about the e-mail, such as the destination address, the sender's address, and the date it was sent.

It also contains a content type that indicates what type of information the e-mail contains. The content type uses the Multipurpose Internet Mail Extensions, or the MIME standard, to specify the type of data. This is the same standard used by the Web servers to send different types of files to your browser.

A normal message has a MIME type of "text/plain". The "text/plain" tells your e-mail client that the e-mail is a plain text message. An HTML message has a MIME type of "text/html". When an e-mail client that recognizes HTML sees this MIME type, it knows that it should display this message as HTML. If the client doesn't recognize the type, it may display it as text with all the HTML tags exposed for your viewing pleasure, or it may not display it at all. The only difference between an e-mail message sent with the CFMAIL tag with the HTML type specified and one sent without it is the content type in the e-mail header. Figure 1 shows a sample HTML e-mail message.

If we want to send a single e-mail message with both HTML and text portions, we need to use a special MIME type of "multipart/alternative." This MIME type allows us to send messages with multiple messages or parts. The "alternative" part of MIME type tells the client to display the best format of the parts that are sent.

In HTML-aware e-mail clients, HTML is considered better than plain text. If you really want all the nitty-gritty details, read RFC 2046. Along with the "multipart/alternative" type, we need to define a boundary. A boundary is a unique series of characters used to separate the different parts of our message.

The boundary can be whatever you want, but you need to make sure it's something that won't appear in the body of the message. If someone were to send a message with the boundary in the text, the e-mail client will improperly break the message into a new part where the boundary text occurs.

```
Date: Sun, 03 Jun 2001 20:58:20 -0400
From: kelly@aboutweb.com
Subject: Multipart Email
To: audience@yourcompany.com
Content-type: multipart/alternative;
              boundary="==MuLtIpArT
BoUnDaRy=="

--==MuLtIpArT BoUnDaRy==
Content-Type: text/plain

This is the text portion of the email.

--==MuLtIpArT BoUnDaRy==
Content-Type: text/html

<body>
<B>
This is the html portion of the email.
</B>
</body>

--==MuLtIpArT BoUnDaRy==--
```

FIGURE 2: A sample multipart e-mail

Figure 2 shows a sample multipart e-mail. We're using a boundary that consists of the text "==MuLtIpArT BoUnDaRy==" with equal signs and lower- and uppercase letters. You're not likely to use this combination of characters in an e-mail message unless you happen to be discussing this article. Although the boundary is on the line after the content type, it's really a part of it.

Notice the semicolon after the MIME type. We leave a blank line after our header and begin the body of our message. We use two dashes to indicate the beginning of the boundary for the first part of our message, then a content type statement followed by a blank line. The boundary and the content type function like the header at the top of the e-mail message, but only for this subpart of the e-mail.

Our first part is the plain text version of our e-mail. After the text version we start the HTML version by putting in a new boundary. Once again we precede the boundary text with two dashes and specify a new MIME type, "text/html". We put in our HTML message using any HTML tags we want. We then close off the second part of our message with the boundary. It has two dashes on the end indicating this was the last part in our message.

The ColdFusion Code

Now we know what the e-mail is supposed to look like, but we have to get ColdFusion to send it in the right format. Listing 1 shows how to do this. First we get the HTML and text version of our e-mail. In the example I use the CFHTTP tag to retrieve the text and HTML versions of the message and store them in the variables txtmessage and htmlmessage. You could also get your content from an HTML form or retrieve it from a database.

Next we need to get the right content type into the e-mail message. The only option for the type parameter in CFMAIL is "HTML" so we need another way to set the content type. Fortunately, Allaire introduced a new tag in ColdFusion 4.5, CFMAILPARAM. The CFMAILPARAM tag is used within the CFMAIL tag and allows you to add mail headers and attach multiple files to an e-mail. We're interested in adding a content type header. The syntax is as follows:

```
<CFMAILPARAM NAME="Header Name"
VALUE="header value">
```

Our header name is "Content-Type" and the value is "multipart/alternative". We also need to include our boundary in the value. In the sample code I placed the boundary in a variable because we'll be using it in several locations. Here's what our CFMAILPARAM statement looks like.

```
<CFMAILPARAM
   NAME="Content-Type"
   VALUE='multipart/alternative;
 boundary="#boundary#"'>
```

After the CFMAILPARAM tag, we start the body of our message. We put the boundary in preceded by the two dashes as in the example. This is where it's handy to put the boundary in a variable. It prevents you from retyping a long string or cutting and pasting and making a mistake. The boundaries must match exactly. We then specify our MIME type and leave a blank line. Since our text message is in a variable, we add it and move on to the next part. We put our boundary in again, this time specifying the HTML MIME type. We output the HTML version of our message and then close out the e-mail message with another boundary, followed by the two dashes to complete our e-mail.

Using this method you can support both HTML-aware and text-only e-mail clients with a single message. However, this method doesn't work for all e-mail clients. Some older clients may not understand the multipart e-mail. For instance, there are still UNIX computers around using e-mail clients like elm that could cause you problems.

Sending An Effective HTML E-Mail

Now that you know how to send an HTML e-mail, make sure you send effective ones. In general, follow the same guidelines that you do for creating well-designed Web pages, but with a few caveats:

- ***All your URLs should be fully qualified:*** You can't use relative links in an e-mail. You must have the full path for all links and images you use in a message.
- ***Users may read their e-mail offline, particularly on services like AOL:*** If users read the message offline, any images you have linked in won't show up. Make sure you use alternate tags and ensure that your message can stand alone without the pictures.
- ***The height and width of the window in which users read their e-mail is usually smaller than when surfing the Web:*** Don't send messages that are too wide. Try to send pages that adjust to a user's window size.
- ***Test, test, test:*** When you put up a Web page you can fix any errors you find later, but once an e-mail is sent it's gone. A broken image or a poor design can make your impressive e-mail backfire and make you look incompetent.

Follow these guidelines to HTML e-mail success. Use the multipart e-mails and keep in mind the differences between HTML pages and HTML e-mails and you'll be able to increase the impact of your e-mail advertising efforts.

ABOUT THE AUTHOR

Kelly Brown is the CTO of About Web (www.aboutweb.com), an Internet solutions provider in the Washington, DC, area. He has a BS and MS in computer science and is a Microsoft-certified systems engineer.

KBROWN@ABOUTWEB.COM

Listing 1

```
<html>
<head>
 <title>HTML Email</title>
</head>

<body>
Sending html email.<BR>

<!--- Get text message --->
<cfhttp
   url="http://127.0.0.1/samplemail.txt"
   method="GET">
</cfhttp>
<CFSET txtmessage=CFHTTP.FileContent>

<!--- Get html message --->
<cfhttp
   url="http://127.0.0.1/samplemail.htm"
   method="GET">
</cfhttp>
<CFSET htmlmessage=CFHTTP.FileContent>

<!--- Define a unique boundary string --->
<CFSET boundary="==MuLtIpArT BoUnDaRy==">

<!--- Send the email --->
<cfmail to="audience@yourcompany.com"
        from="kbrown@aboutweb.com"
        subject="HTML Email">
<CFMAILPARAM
   NAME="Content-Type"
   VALUE='multipart/alternative;
 boundary="#boundary#"'>

--#boundary#
Content-Type: text/plain

#txtmessage#

--#boundary#
Content-Type: text/html

#htmlmessage#

--#boundary#--
</cfmail>
</body>
</html>
```

CODE LISTING

The code listing for this article is also located at

www.sys-con.com/coldfusion/sourcec.cfm

CFDJ FEATURE

BY CURTIS SCHLAK

Wizards on the Web

Techno-evangelists have been proclaiming for years that the Internet will change the world, bring people closer, and provide the ultimate forum for sharing information and exchanging opinions. Yet something as simple as filling out a Web-based form remains, in large part, about as much fun as filling out your tax forms with a crayon.

A good option when Web surfers are required to input information is a wizard. Wizards make it easy for Web surfers to input information into Web-based forms, particularly when that information is complex, lengthy, or when it varies based on the user who is providing it. Wizards are more intuitive than typical HTML-based forms, and ColdFusion's functionality gives programmers tremendous flexibility when it comes to manipulating the information into the ColdFusion database.

In this article we'll build the code that enables a wizard to enter a recipe – a seemingly harmless task that, without a wizard, becomes devastatingly complex for the developer and the user. This is a simplified version of a recipe application that I built in 1998. The code may seem like overkill, but when faced with a six-step process, it becomes invaluable.

Purpose of a Wizard

In the early days of Web-based forms (we're talking 1997 here), surfers would often scroll through forms up to eight screens long

before finding a "Submit" button. Someone finally got the bright idea to break that form up into multiple pages. The concept of the Web-based wizard was born.

Sectioning related groups of requested information onto separate pages gives the user a better experience. Every modern e-commerce checkout process involves a wizard. Signing up for a free magazine on the Web usually involves a wizard. Whenever you come across the words, "Step 3 of 5," you're in a wizard.

Advantages of Using Wizards on the Web

The most obvious reason to use a wizard is to chop up that long form you've created. But there are subtler reasons to do this. You can lead the user through complicated data input that would be frustrating or confusing to do on a single, long page.

Disadvantages of Using Wizards

There is one disadvantage to doing anything over the Web: slow Internet access. You know the site: some designer thought it would be groovy to fill the home page with 150Kb of images because they thought it would be pretty. Unusable. Undesirable. Unacceptable.

Thirty-five million Web surfers now access the Internet at 56Kbps or less. A little more than 5% of consumers have access to DSL, cable, or other high-bandwidth connectivity. The number of modem connections to the Web is expected to increase through 2004. We, as developers, must take it all into consideration when writing.

If it takes longer for a page to download than it takes the user to fill out requested information, the person gripping the mouse is not going to be happy. Planning the wizard well will minimize the problems that Joe Q. Surfer will have on his WebTV console.

Planning the Wizard

We're on the Web. We've got to get the information that someone has put into a form, and then has hit the "Submit" button. Now what?

Where Am I?

It's a fundamental question. I'm in my living room, having just finished watching another episode of *Law & Order*, which always inspires me to write efficient code. In coding, as in life, it's always a good thing to know where you are.

I use a variable named CurrentStep throughout my wizards. It lets my application know what page the user is coming from and what information to expect. Sometimes it's a hidden form field. Sometimes it's in the URL. The structure of delivery doesn't matter since ColdFusion is smarter than your average application server. I just need to let my application know what to expect and when.

Table 1 represents the recipe wizard's different steps and the information collected from each one.

We'll use this information to make our application smart about catching information.

Step	Information
1	Name Description (not required) Copyright (not required) Owner (not required) Public Daypart Food Type (not required)
2	Procedure (not requied)
3	Ingredient Amount
4	Confirm recipe
5	Insert into database

TABLE 1 Steps in a recipe wizard

Building a Structure

ColdFusion structures are a perfect place to put information. If you're unfamiliar with structures, refer to www.allaire.com/documents/cf4/Advanced_ColdFusion_Development/06_Working_with_Structures/adv06.htm for some documentation. Since we're building a wizard that must collect information and maintain it in an organized manner, the structure is the perfect storage type.

Our structure will have the form and recipe information shown in Table 2 stored in it.

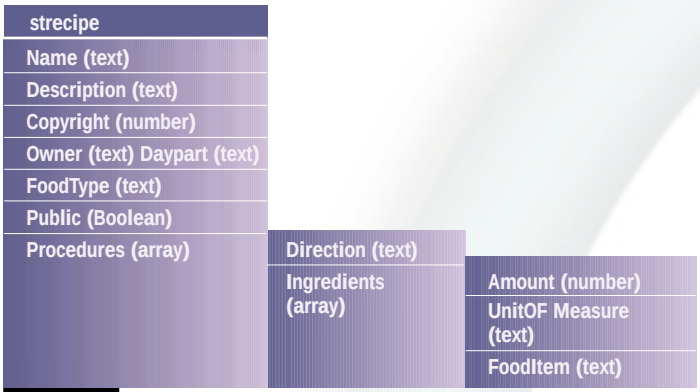


TABLE 2 ColdFusion structure for recipe wizard

ColdFusion allows two syntaxes to access information within a structure: stStructureName["KeyName"] and stStructureName.KeyName. You can mix and match the syntaxes as well. To access the food item in the second ingredient of the fourth procedure, I could use any of the following:

```
stRecipe[ "Procedure" ][ 4 ][ "Ingredients" ][ 2 ][
"FoodItem"]
stRecipe.Procedure[ 4 ].Ingredients[ 2 ].FoodItem
stRecipe[ "Procedure" ][ 4 ].Ingredients[ 2 ][
"FoodItem"]
```

Throughout this article I'll use the square bracket notation, unless I'm referring to the Application, Session, Client, Attributes, or Caller scopes. A typical Session variable that contains a structure would look like this:

```
Session.stMyStructure[ "Akey" ]
```

Now that we've got a structure planned, we can start building code to populate it.

Storing the Structure

State management is a discussion that could take a whole other article. Since we must maintain some state between page requests, so that we can accurately reference information in memory, this discussion is necessary. I'm just going to hit the top-level pros and cons here. I'm using Session variables in this example, but any of these are available.

Session Variables

Session variables time out. They rely on a CFID and CFTOKEN (autogenerated by ColdFusion Server) combination to uniquely identify a Session. They're really convenient. ColdFusion 4.0+ requires a lock on accessing these variables if your application runs in a multi-threaded environment. This incurs a small performance hit.

Client Variables

Like Session variables, Client variables rely on a CFID and CFTOKEN combination to identify a Client. On the plus side, you can store them in cookies, in a database, or (forbid!) the system registry. They don't time out, unless the system wipes them clean after being in disuse for a while.

Form Fields

Passing information along in a series of hidden form fields can be good at times, but they increase the download time for a page. They are hard to maintain and are useless without a form submission. This is one of the last options I would ever consider using.

URL Variables

As a developer, I get excited about clean, efficient code. When I look at a URL that is choked with encoded variables, it turns me off. Here's an example of one that makes my hair stand on end:

http://www.askjeeves.com/main/metaanswer.asp?metaEngine=directhit&origin=0&MetaURL=http%3A%2F%2Fask%2Edirecthit%2Ecom%2Ffcgi%2Dbin%2FRedirURL%2Efcg%3Furl%3Dhttp%3A%2F%2Fdetlefmp%2Ecom%2Fwwwboard%2Fmessages%2F12826%2Ehtml%26qry%3DWhy%2Bwould%2BI%2Bever%2Bwant%2Bto%2Buse%2Ba%2Bthousand%2Burl%2Bvariables%253F%26rnk%3D1%26src%3DDH%5FAsk%5FSRCH&qCategory=ref_&metaTopic=Get+Thousands+of+Visitors%21+Use+OUR+PROMOTIONAL+TOOL%21%2A%2A%2AClick+HErE&ItemOrdinal=0&logQID=67E4EB0AA84366488C4BFAC43559D351&sv=5&back=http%3A%2F%2Fwww%2Easkjeeves%2Ecom%2Fmain%2Faskjeeves%2Easp%3Fask%3DWhy%2Bwould%2BI%2Bever%2Bwant%2Bto%2Buse%2Ba%2Bthousand%2Burl%2Bvariables%253F%26origin%3D0%26site%5Fname%3DJeeves%26metasearch%3Dyes%26ads%3D

Yikes. Since I am an aesthetic programmer, if I were ever to use this method, I would hide the URL from the user with a frameset and use the pages in the frame to accept the egregious URL. The user, meanwhile, sees only the www.yourdomain.com in the address bar.

Database

It's conceivable that a programmer could store and retrieve information from a database with every form submission. It is persistent and remains safe in the warm arms of relational tables. Still, when I use a database connection to pump that information back and forth, I get worried. What if, for some reason, the database is being hit really hard? I don't know, I've got reservations about it. It remains an option, but my preference is to store stuff in Session or Client variables, if robustness (or whatever) is critical.

File

One of the most interesting ColdFusion developers I know uses a file to store a WDDX packet of serialized ColdFusion structures. I've done it, too, and it works well. Unfortunately, this option draws the ire of UNIX administrators when these files begin clogging their directories. On a Windows PC, I have less faith in the file access, not from any proof other than the number of corrupted files I've come across on my NT boxes. Again, I prefer to stick with Session or Client variables stored in a database.

Using Application.cfm

I firmly believe that a ColdFusion developer should never choke the Application.cfm with a bunch of silly stuff. Normally

I would consider form handling to be a silly thing. In the case of wizards, however, our application must be ready to receive information between any two points of the process. The most logical place, then, is to store the data in a local Application.cfm.

For the people asking themselves, "What is this Application.cfm that he keeps mumbling about?" please allow me to explain.

ColdFusion has two templates that run every time a ColdFusion page is requested. They are Application.cfm and OnRequest-End.cfm. Application.cfm runs before a ColdFusion page and OnRequestEnd.cfm runs after a ColdFusion page. Most people know about Application.cfm. If you never heard of it, don't admit to it. It's one of those pieces of information that if you don't have, Allaire will strip you of your ColdFusion T-shirt. A lot of people don't know about OnRequestEnd.cfm. I like to put Easter eggs in mine. It enlivens the application with a little chance.

Since the Application.cfm is always willing to run before any page in a wizard, it's the perfect place to put form-handling code. The application captures form information as it inputs. This is important because it allows the form to save information in the event the user hits the "Back" or "Previous" button.

Handling the Form Collection

There are two types of fields in a form submittal – fields that always appear for your form-handling functions and fields that sometimes show up for those functions. Text areas and boxes, selects, and hidden fields show up for the functions to handle all the time. If these fields are in a form and you hit the "Submit" button, the information will show up for the functions to handle, even if they contain empty strings.

Regardless of whether the fields appear in the posted page, I perform some server-side validation. If there are any field values that do not comply with my rules, I append that field name to a list of bad field names, reload the appropriate form, and mark the erred entries, usually by changing the attributes of the form field's label. For example, if there is incorrect or incompatible information entered into a field named *Age*, the following code would change the text label for that field to show that there was an error – by making the word all capital letters, red, and bold.

```
<td>
 <cfif IsDefined( "lBadFields" )
AND ListFindNoCase( lBadFields, "Age" )>
  <font color="red"><b>AGE</b></font>
 <cfelse>
  Age
 </cfif>
</td>
```

Checkboxes and radio buttons, on the other hand, don't show up reliably if they're not checked. This can complicate an application. Sometimes, form-field functions overlap. I use the following strategies for handling form data in a wizard.

Checkboxes and Radio Buttons

Since checkboxes and radio buttons must be handled independently of their existence in a submitted form, I use the CurrentStep value to help my application create the appropriate values.

On the first step the Daypart, Food type, and Public are checkboxes. I am expecting three form fields: Form["Daypart"], Form["FoodType"] and Form["Public"]. If someone chooses not to check any of those boxes, none of these form fields appear in the posted page. So, for every checkbox and radio button in the step, I'll write a special handler for it. The code in Listing 1 would handle the Daypart group.

Text and Select Fields

All the remaining fields can be garnered by having a little fun with the <cfloop> tag. Here's an example of capturing the information from Step 3 in Table 1, the Ingredient item, amount, and unit of measurement. This will add an ingredient to the end of the list. Passed with the form is a hidden field (Form["ProcedureNumber"]) that tells stores the Procedure number (see Listing 2).

Now, to whatever page the form is submitted, the structure will be able to get all the required information that it needs.

Mutually Exclusive Fields

There are times when you will have two form fields that, if you have a value in one, the other should not be considered. If there was a unit of measurement dropdown with a field name UnitOfMeasurementSelect, and a text field next to it named UnitOfMeasurementText that would allow a user to type in a custom unit of measurement, then the server must take into account that a custom value supersedes a predefined value. I recommend handling that with JavaScript, but, if you're in an all server-side validation shop, then you could use something like the code shown in Listing 3.

Multidirectional Wizard Navigation

A wizard should allow the user to backtrack without losing any information. The code techniques developed above allow form information to be captured within the wizard. Now, how do we get that information to the server, regardless of the user's traveling direction?

Remember the variable CurrentStep? The form validation can result in two states: "everything's okay" and "there's something wrong." If there's something wrong, we don't want to move away from the current step of the wizard. If everything is okay and we're going forward, then we add 1 to CurrentStep. If everything is okay and we're going backward, then we subtract 1 from CurrentStep. If we adopt the page design shown in Figure 1, then we can always include the correct page.

Somewhere at the top of RecipeWizard.cfm, I stick the following code:

```
<cflock type="READONLY" scope="SESSION" timeout="30">
 <cfset stRecipe = StructCopy( Session.stRecipe )>
</cflock>
```

This creates a local copy of the recipe structure that my application can use without incurring the performance hit of a lock on the Session scope every time I need to reference a value.

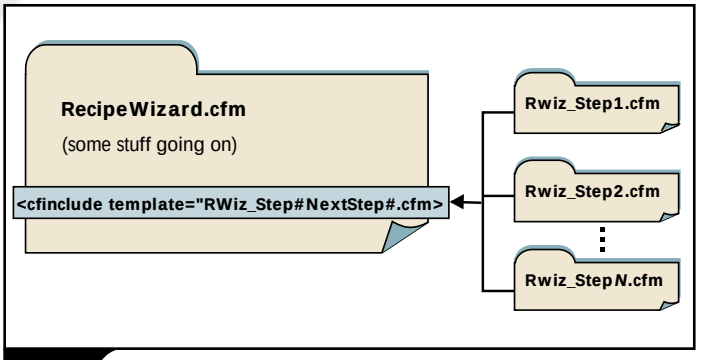


FIGURE 1: CurrentStep variable

ColdFusion fasttrack

The ONLY ColdFusion Event Backed by the Power of SYS-CON MEDIA and ColdFusion Developer's Journal

ColdFusion fasttrack

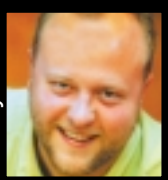
REGISTER ONLINE TODAY! Save $300

Save the Dates

Prepare for ColdFusion Certification

Kevin Lynch

Kevin Lynch is president of Macromedia Products. He joined Macromedia in 1996 and has been instrumental in forming its Web strategy. As president of products, Kevin is responsible for developing Macromedia's award-winning family of software and solutions.



Jeremy Allaire

As Chief Technology Officer, Jeremy is instrumental in guiding Macromedia's product direction and is the company's primary technology evangelist, responsible for establishing key strategic partnerships within the Internet industry. Jeremy was the founder and former chief technology officer for Allaire Corporation, which merged with Macromedia in March 2001.

Ben Forta

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 its sequel,* Advanced ColdFusion 4.0 Development, *as well as* Allaire Spectra E-Business Construction Kit *and* Sams Teach Yourself SQL in 10 Minutes. *He recently released* WAP Development with WML *and* WMLScript.

Prepare for the ColdFusion Certification Exam with Comprehensive Technical Sessions Designed for Beginner and Advanced ColdFusion Developers

Session Topics Include:

- Web Fundamentals
- Application Development
- Database Concepts
- Client State Management
- Troubleshooting, and A Special Focus on ColdFusion 5.0.

Plan to Attend

webServices JOURNAL | JAVA DEVELOPER'S JOURNAL | XML JOURNAL | wireless | LINUX BUSINESS WEEK | ColdFusion Developer's Journal | WebSphere DEVELOPER'S JOURNAL | Dr. Dobb's | webtechniques

Sponsored by

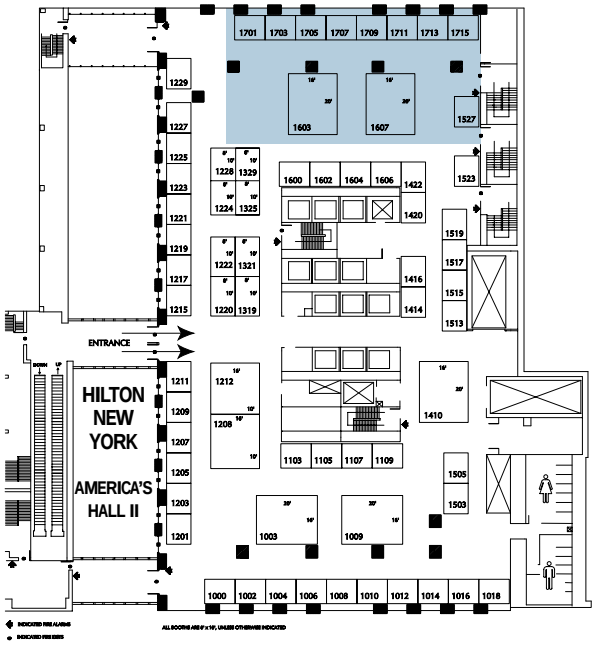
CF Advisor & ColdFusion Developer's Journal

HILTON NEW YORK, NYC, SEPTEMBER 24-25

COLDFUSION PAVILION

Hilton New York, NYC
September 24-25

Exhibit at the "COLDFUSION PAVILION"



Supported by a **Macromedia ColdFusion FastTrack & Certification Program,** the **"CF Exhibit Pavilion"** will be sponsored by **SYS-CON Media's** market-leading **ColdFusion** publication, ***ColdFusion Developer's Journal.***

Don't miss your best and only opportunity to reach **ColdFusion** experts and just-certified professionals.

Join ***ColdFusion Developer's Journal*** and bring your **ColdFusion** products, services, and solutions to **3,000+ business** and **technology professionals.**

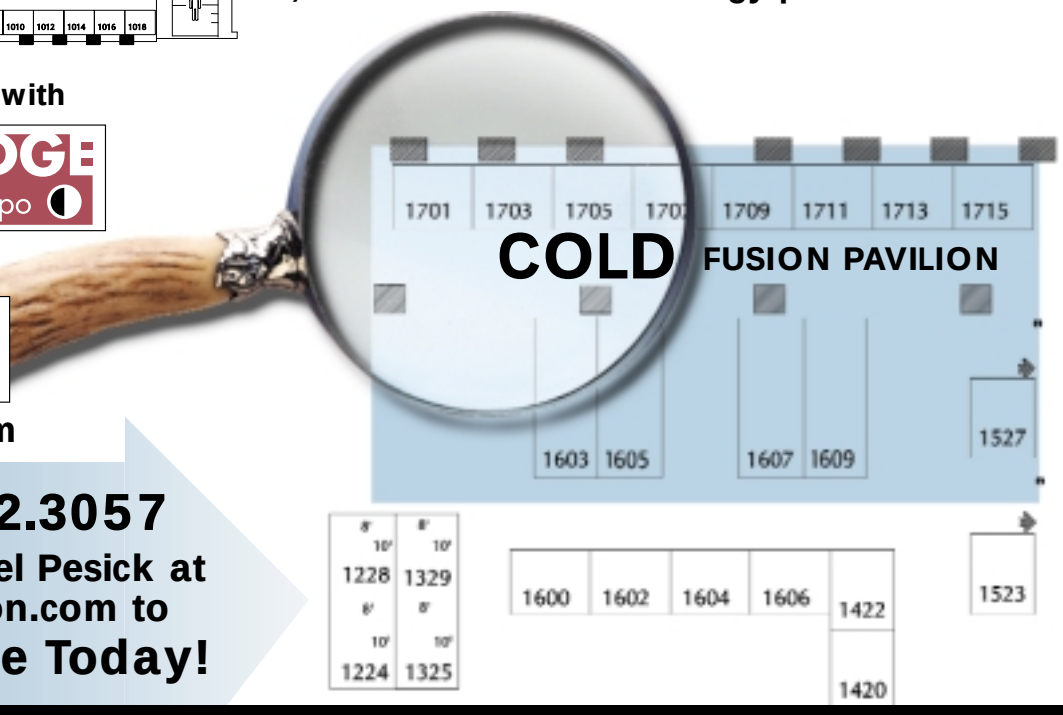
Colocated with

JDJ EDGE conference&expo

&

web services EDGE conference&expo

www.sys-con.com



Call 201802.3057

or e-mail Michael Pesick at michael@sys-con.com to

Reserve Space Today!

Owned and Produced by

SYS-CON EVENTS

135 Chestnut Ridge Road, Montvale, NJ 07645
(201) 802-3069 Fax: (201) 782-9051
visit
WWW.SYS-CON.COM
for more information

JavaScript Modification

Netscape Navigator 2+ and Microsoft Internet Explorer 4+ allow developers to read and write to the action property of a form. I usually set the form's action property to <cfoutput> #CGI.SCRIPT_NAME#</cfoutput> so that the form just submits back to the same page. (In case you don't know about the CGI variables, look into them. They are amazing, at times.) Now, when the forward button is hit, the default action is produced. If a user clicks the "Go Back a Step" button, then we have to do something to allow the application to know where the user wants to go.

I solve this by appending a URL variable to the action before submission. The button shown in Listing 4 would do the job.

All that's left is the code in Application.cfm to figure out where to go. That code could look something like Listing 5.

Combining this with the page design and the <cfinclude template="RWiz_Step#NextStep#.cfm">, we are assured that the user can navigate easily through the wizard while we continue to collect all the required information.

Populating Fields

We've spent all this time collecting information intelligently from forms and allowing our user to traipse all over the wizard willy-nilly. The last action is to display any information that we may have stored. This prevents users from being confused by empty fields that they thought they had filled out. Luckily, we already have the information stored in an intuitive and easily accessible manner.

If the user comes back to the first page sometime during his or her experience, we can populate the fields of the first page with something like:

```
<td>Name</td>
<td>
 <input type="text" name="Name" size="30" maxlength="128"
  <cfif StructKeyExists( stRecipe, "Name" )>
   value="<cfoutput>#stRecipe[ "Name" ]#</cfoutput>"
  </cfif>>
</td>
```

If the structure has a Name stored in it, then it will populate the field with that value.

The Final Step

The last thing to do is put all this information somewhere. Usually it's headed for a database. This structure allows you to do looping over arrays and nested structures to perform the insertion in a mechanical mode.

The concept of a wizard is easy: provide questions in a logical order and in intuitive groupings. This coding technique makes it extremely simple for a programmer to do just that. Next time a client asks you to collect 37 pieces of information from a user, or your application demands that you collect structured information, think about using a wizard. You and your user will be much happier with the result.

About the Author

Curtis Schlak received a BS in mathematics with a minor in English. His first self-published textbook, The Sad Book of Calculus: A Textbook Supplement, *sold out. During a spell in the army, he integrated Microsoft products with legacy dBase logistics software and wrote his first Web application. He was a senior director of development (ColdFusion) at KickFire, Inc., in Saratoga, California. He has now started a new company, Striatum Solutions, in Houston, Texas.*

curtis_schlak@yahoo.com

Listing 1

```
<cfif ( StructKeyExists( Form, "CurrentStep" )
   OR StructKeyExists( URL, "CurrentStep" ) )
   AND CurrentStep EQ 1>
 <cflock type="EXCLUSIVE" scope="SESSION" timeout="30">
  <cfscript>
   // If the Day part group exists, add it to the struc-
ture
   if( StructKeyExists( Form, "DayPart" )
    AND Len( Trim( Form, "DayPart" ) ) )
   {
    Session.stRecipe[ "DayPart" ] = Form[ "DayPart" ];

   // otherwise delete the structure key
   // and add it to the bad field list since
   // it is a required field
   } else {
    StructDelete( Session.stRecipe, "DayPart" );
    lBadFields = ListAppend( lBadFields, "DayPart" );
   }
  </cfscript>
 </cflock>
</cfif>
```

Listing 2

```
<cfset NumberOfIngredients = ArrayLen( Session.stRecipe[
"Procedures" ][ Form[ "ProcedureNumber" ][ "Ingredients" ]
)>
<cfloop collection="#Form#" item="FieldName">
 <cfswitch expression="#FieldName#">
  <!--- This case is to capture required text values --->
  <cfcase value="IngredientItem, other field names">
   <cfif Len( Trim( Form[ FieldName ] ) )>
    <cfset Session.stRecipe[ "Procedures" ][ Form[
"ProcedureNumber" ] ][ "Ingredients" ][ NumberOfIngredients
+ 1 ][ FieldName ] = Form[ FieldName ]>
   <cfelse>
    <cfset temp = StructDelete( Session.stRecipe, FieldName
)>
    <cfset lBadFields = ListAppend( lBadFields, FieldName
)>
   </cfif>
  </cfcase>

  <!--- This case is to capture numeric text values --->
  <cfcase value="IngredientAmount, other field names ">
   <cfif IsNumeric( Form[ FieldName ] )>
    <cfset Session.stRecipe[ "Procedures" ][ Form[
"ProcedureNumber" ] ][ "Ingredients" ][ NumberOfIngredients
+ 1 ][ FieldName ] = Form[ FieldName ]>
   <cfelse>
    <cfset temp = StructDelete( Session.stRecipe, FieldName
)>
    <cfset lBadFields = ListAppend( lBadFields, FieldName
)>
   </cfif>
  </cfcase>

  <!--- This case is to capture optional fields --->
  <cfcase value="UnitOfMeasure, other field names ">
   <cfset Session.stRecipe[ "Procedures" ][ Form[
"ProcedureNumber" ] ][ "Ingredients" ][ NumberOfIngredients
+ 1 ][ FieldName ] = Form[ FieldName ]>
  </cfcase>
 </cfswitch>
</cfloop>
```

Listing 3

```
<cfset NumberOfIngredients = ArrayLen( Session.stRecipe[
"Procedures" ][ Form[ "ProcedureNumber" ][ "Ingredients" ]
)>
<cfloop collection="#Form#" item="FieldName">
<cfif StructKeyExists( Form, "UnitOfMeasurementText" )>
 <!--- Start with the highest priority Form field --->
 <cfif Len( Trim( Form[ "UnitOfMeasurementText" ] ) )>
  <cfset Session.stRecipe[ "Procedures" ][ Form[
"ProcedureNumber" ] ][ "Ingredients" ][ NumberOfIngredients
+ 1 ][ "UnitOfMeasurement" ] = Form[
"UnitOfMeasurementText" ]>

 <!--- Continue with lower priority fields --->
 <cfelseif Len( Trim( Form[ "UnitOfMeasurementSelect" ] )
)>
  <cfset Session.stRecipe[ "Procedures" ][ Form[
"ProcedureNumber" ] ][ "Ingredients" ][ NumberOfIngredients
+ 1 ][ "UnitOfMeasurement" ] = Form["UnitOfMeasurementText"
]>

 <!--- Finally, if it all fails, get rid of any current
value --->
 <!--- and mark it as a bad field
--->
 <cfelse>
  <cfset temp = StructDelete(Session.stRecipe[ "Procedures"
][ Form[ "ProcedureNumber" ] ][ "Ingredients" ][
NumberOfIngredients + 1 ], "UnitOfMeasurement" )>
  <cfset lBadFields = ListAppend( lBadFields,
"UnitOfMeasurement" )>
 </cfif>
</cfif>
```

Listing 4

```
<form action="<cfoutput>#CGI.SCRIPT_NAME#</cfoutput>"
method="post" name="anotherForm">

Stuff

<input type="button" value="Back to Step 2" onclick="docu-
ment.anotherForm.action += '?wizardback=true';
document.anotherForm.submit()">
</form>
```

Listing 5

```
<!--- Check for any invalid form entries                --
->
<!--- If there are some, we need to redo the same page ---
>
<cfif Len( Trim( lBadFields ) )>
 <cfset NextStep = CurrentStep>

<!--- Check to see if the user needs to go back a step ---
>
<cfelseif StructKeyExists( URL, "wizardback" )>
 <cfset NextStep = Min( CurrentStep – 1, 1 )>

<!--- Default to going forward --->
<cfelse>
 <cfset NextStep = CurrentStep + 1>
</cfif>
```

CODE LISTING

The code listing for this article is also located at

www.sys-con.com/coldfusion/sourcec.cfm

ColdFusion **Meets Flash**

The power of integration

BY
DENNIS BALDWIN

Most developers use Flash for animation and Web site introductions, but aren't aware of how powerful a scripting environment it is. Now in Flash 5, ActionScript is larger than life and is here to stay.

It's pretty much a programming language with access to variables, expressions, arrays, objects, and user-defined functions. ActionScript is very similar to JavaScript and both are based on the ECMA-262 specification.

Developers rave about how great ColdFusion is for rapidly developing e-commerce sites and its ease of use. Flash advocates can't get over its small file sizes, cross-browser compatibility, and Flash player domination throughout the Web. Wouldn't it be great to integrate the two! We would end up with a powerful, server-side technology that does all the dirty work along with a client-side app that looks great.

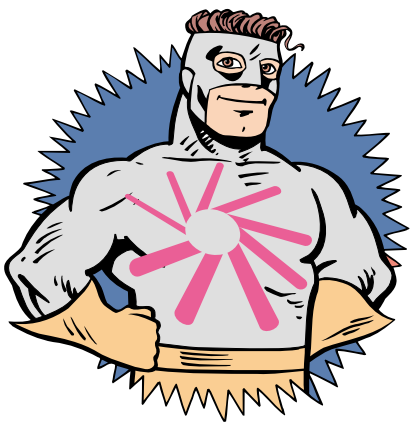
This article demonstrates how to integrate ColdFusion and Flash to create a dynamic navigation menu. (Source files for this article can be found on the ***CFDJ*** Web site, www.sys-con/coldfusion/sourcec.cfm.) The process basically goes as follows:

1. Flash loads variables from ColdFusion.
2. ColdFusion queries databases and outputs variables to a template.
3. Flash checks to see that the variables are fully loaded and then parses them.
4. Movie clips are duplicated and variables are set to create the navigation menu.

The Server Side

For simplicity, we'll use an Access database (flashmenu.mdb) for this tutorial. Pretty much any database that ColdFusion can talk to can be used. We'll be dealing with two database tables: "cats" and "subcats." The structure of these tables is fairly simple.

The CatID field is the primary key for the cats table and the foreign key for the subcats table. This allows us to link the subcategories that correspond with the main categories. Once the data structure has been developed, we can create the DSN. In our example we name the DSN "flashmenu", but feel free to use whatever you prefer. Just make sure you change this reference in the ColdFusion template (cats.cfm). Let's look at the cats.cfm template. Flash is picky about white space and carriage returns, so it's good practice to use the <CFCONTENT> and <CFSETTING> tags.



```
<cfcontent type="text/plain">
<cfsetting enablecfoutputonly="yes">
```

This will set the MIME type of the template to text and eliminate any unnecessary white space. We'll then query our database and grab the categories from the cats table.

```
<!--- query to get the main categories --->
<cfquery datasource="flashmenu" name="get_cats" dbtype="odbc">
select catid,category
from cats
</cfquery>
```

A few variables need to be initialized so we can use them in our loop. We create a count variable that will be used to track our current index in the loop. We also create an empty container for the categories that store in a pipe-delimited list. You can specify anything for the delimiter, but I prefer using the pipe symbol. In most cases it doesn't conflict with any of the data in the list.

```
<!--- initialize the category list --->
<cfset cats="">
<!--- initialize the counter --->
<cfset count=0>
```

Now we can loop through our initial "get_cats" query and create a subquery that will grab our subcategories based on our foreign key – CatID. This allows us to join our data so that CatID 1 will pull all the subcategories that fall under this category ID. We're now able to start filling our cats container with data that will be used in Flash (see Listing 1).

Now that we've taken care of the main categories we need to start building the subcategory lists. To do this we need to initialize a few more variables and create a subloop that will loop through the "get_subcats" query (see Listing 2).

We've now established all the necessary lists that we need: cats, subcats, and URLs. This data will be used shortly, but first we need to output our information to the page (see Listing 3).

We use the loaded variable to check inside Flash that all the variables are fully loaded. You can see the output of this template by hitting the "cats.cfm" page through your Web browser. Everything is in place on the ColdFusion end; now we need to load the information from our template into Flash.

MACROMEDIA

WWW.MACROMEDIA.COM/DOWNLOADS

The Client Side

Now that we've tackled the server code we'll look into the Flash end of things. "Flashmenu.fla" is the Flash source file we'll be dealing with in this tutorial. Our file mainly consists of two movie clips: "cat" and "subcat." You can view all symbols used in our Flash movie by going to Window>Library or using "Ctrl+L". This displays the Flash library for the current movie. You'll see two folders that contain all of the components used in our navigation menu. There are two layers in our movie; the top layer "AS" contains the ActionScript and the bottom layer "cats" holds the movie clip that we'll duplicate to create the categories in the menu.

Let's look at the code in frame 1 of the AS layer. You can activate the ActionScript window by double-clicking the frame, using the keyboard shortcut "Ctrl+Alt+A", or by going to Window>Actions. In the first frame you'll see the following code:

```
// Load the variables from
ColdFusion
loadVariablesNum ("cats.cfm", 0);
// set the initial cat clip
visibility to 0
cats_clip._visible=0;
// initialize the selected
variable
selected="default";
```

You'll see that we load the variables from the ColdFusion template we created earlier. One important thing to note about loading variables from CF: if you would like to test inside of Flash make sure to use an absolute URL:

```
// Load the variables from
ColdFusion
loadVariablesNum
("http://localhost/cats.cfm", 0);
```

This saves a lot of time when developing an application because you don't have to publish the movie every time and test it. It also helps for debugging purposes so you can use the Flash debugger and output your variables to the window. Once you have created an absolute path to your template, you can test the movie by going to Control>Test Movie or using "Ctrl+Enter". If you have configured everything correctly, the navigation menu should display with all the categories and subcategories. When a category is selected, the menu will expand so you can select a subcategory. When a subcategory is clicked, you'll be sent to the corresponding Web site.

Let's take a closer look into how the menu is dynamically created. First, the ActionScript in frame 3 looks like:

```
if (loaded == 1) {
gotoAndStop (4);
} else {
gotoAndPlay (2);
}
```

When the LoadVariables command is used we don't immediately have access to the variables. Frame 3 tests to see if the "loaded" variable is equal to 1. If so, we'll go to frame 4 and stop or go back to frame 2 and keep looping until the "loaded" variable equals 1. Remember when we created the ColdFusion template we placed this variable last. This means that if Flash sees this variable, then everything has completely loaded. Frame 4 might be a bit intimidating at first, but a simple explanation should clear everything up (see Listing 4).

Our ColdFusion variables are now fully accessible to us in the main timeline. So the first thing we do is take the "cats" variable and split in into an array. This will allow us to have easy access to the elements in the array, which you'll see shortly. We then set a "ystart" variable that gets the *y* position of the main cat clip. This will be our reference point when we begin duplicating the movie clips. No matter where we place the initial clip on the stage, the "ystart" variable will be dynamic and our menu will be built correctly. Also, movie clips are referenced through an instance name, in this case "cats_clip". You can set the instance name by accessing the instance panel under Window>Panels>Instance or using "Ctrl+I". Each movie clip can have its own instance name with its own set of properties. This is one reason Flash is so powerful, we can create a symbol one time and use it over and over without drastically increasing file size.

We now loop through the categories, duplicate the "cats_clip", and set the category names. Inside each category movie clip exists a clip called "subcat_container". This container clip holds the subcategory movie clip and the ActionScript that duplicates the subcategory clips. The ActionScript in this clip is similar to the previous code snippet.

We need to consider what happens when the user clicks on a category or subcategory. Inside the "cats_clip" is a button containing the ActionScript that controls the movie clips when a category is selected. When a category is clicked, the selected clip that displays the subcategories will play. The other clips are repositioned along the *y* axis to accommodate this. When a subcategory is clicked, the URL loaded from ColdFusion is targeted and the user is sent to that Web site.

Conclusion

The power of this tutorial lies in how dynamic the menu is. If a new category and subcategory are added to the database, the Flash file is automatically updated with the new information. All clips will reposition accordingly and everything should be in working order.

There are a few things to consider when implementing this in a production environment. This tutorial was developed to demonstrate the power of integrating ColdFusion and Flash. It's not very efficient to query the database for categories and subcategories every time the page is loaded. This information might not be frequently updated, therefore query caching might be a good idea. Another suggestion is to set a scheduled task every night to query the database and write the output to a text file. This text file can be loaded into Flash and will reduce the load on the server.

If you have any questions about this article or the source files please feel free to contact me at dennis@zapdesigns.com. For more information on integrating ColdFusion and Flash, check out www.flashcfm.com.

ABOUT THE AUTHOR

Dennis Baldwin is a developer and cofounder of ZAP Designs, Inc., an Allaire Consulting Partner specializing in Flash and ColdFusion development. Among other things, Dennis helps develop and maintain an online resource for Flash and ColdFusion developers – www.FlashCFM.com.

DENNIS@ZAPDESIGNS.COM

Listing 1

```
<!--- loop through the categories query --->
<cfloop query="get_cats">
<!--- append the category to the category list --->
<cfset cats=ListAppend(cats,category,"|")>
<!--- now query to get the sub categories and url links --->
<cfquery datasource="flashmenu" name="get_subcats" dbtype="odbc">
select subcategory,url
from subcats
where catid=#catid#
</cfquery>
```

Listing 2

```
<!--- initialize the subcats list --->
<cfset subcats="">
<!--- initialize the urls list --->
<cfset urls="">
<!--- create another loop for the sub categories --->
<cfloop query="get_subcats">
<cfset subcats=ListAppend(subcats,subcategory,"|")>
<cfset urls=ListAppend(urls,url,"|")>
<!--- end sub categories loop --->
</cfloop>
```

Listing 3

```
<!--- output the subcats and the urls --->

<cfoutput>&subcats#count#=#subcats#&urls#count#=#urls#</cfoutput>
<cfset count=count+1>
<!--- end categories loop --->
</cfloop>
<!--- output the cats --->
<cfoutput>&cats=#cats#&loaded=1&</cfoutput>
```

Listing 4

```
// take the cats variable and split it into an array
cats = cats.split("|");
// get the initial cats clip y position
ystart = getProperty("cats_clip", _y);
// create the loop to duplicate the cat clips
for (i=0; i<cats.length; i++) {
     duplicateMovieClip ("cats_clip", "cats_clip"+i, i);
     setProperty ("cats_clip"+i, _y, ystart+i*20);
     // set the category text in the cat clip
     set ("cats_clip"+i+".category", cats[i]);
     // set the sub category index in the subcat container
     set ("cats_clip"+i+".subcat_container.index", i);
     // set the sub category variable in the subcat container
     set ("cats_clip"+i+".subcat_container.subcat", eval("subcats"+i));
     // set the url variable in the subcat container
     set ("cats_clip"+i+".subcat_container.url", eval("urls"+i));
}
```

CODE LISTING

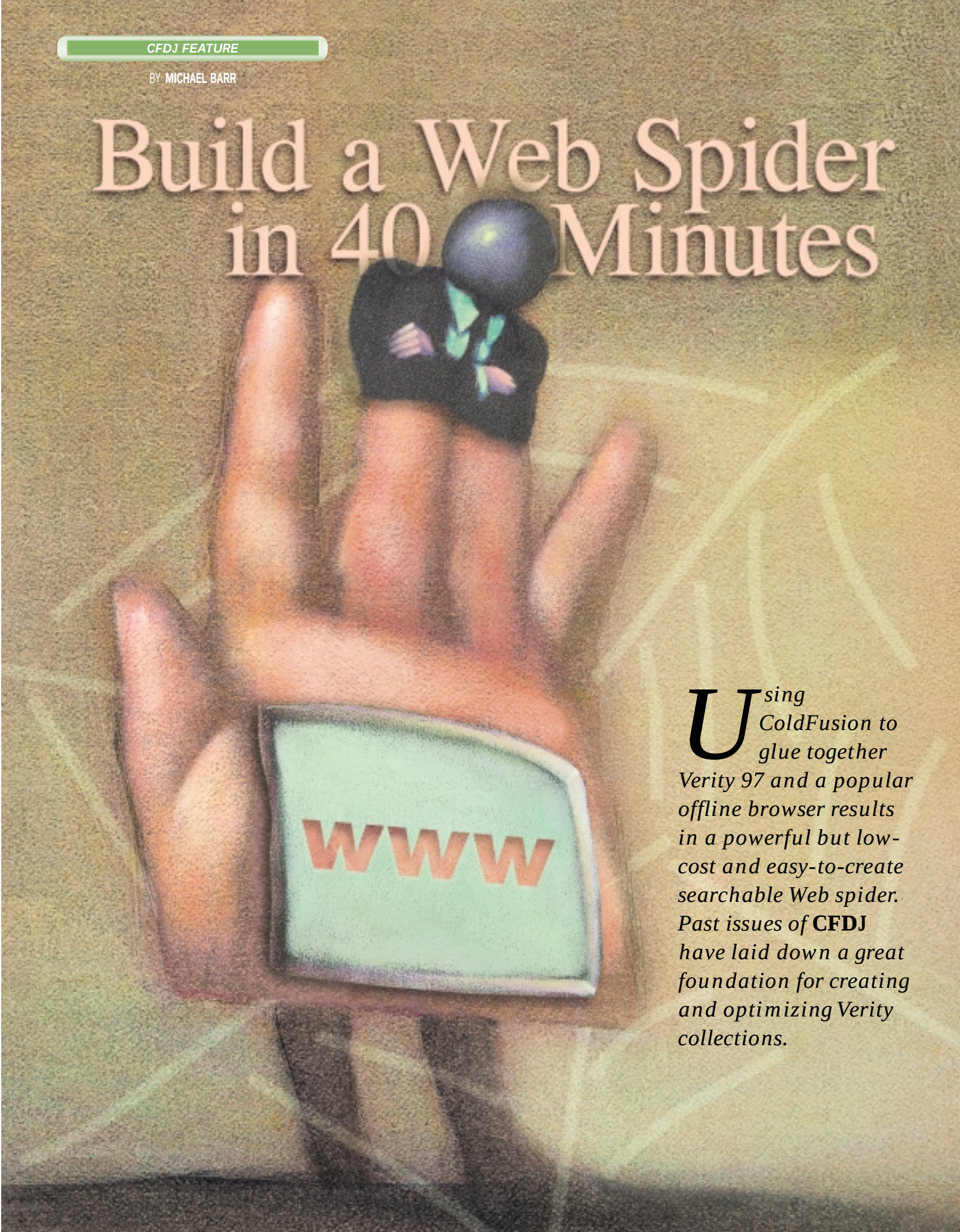
The code listing for this article is also located at

www.sys-con.com/coldfusion/sourcec.cfm

CFDJ FEATURE

BY MICHAEL BARR

Build a Web Spider in 40 Minutes



Using ColdFusion to glue together Verity 97 and a popular offline browser results in a powerful but low-cost and easy-to-create searchable Web spider. Past issues of **CFDJ** *have laid down a great foundation for creating and optimizing Verity collections.*

A starting point for creating your own low-cost, searchable Web spider

This article expands on ***CFDJ***'s Verity-related articles by demonstrating how to create an Internet Web spider not unlike Yahoo. Since Verity and the offline browser utility do most of the work, simple ColdFusion code is used to create the searchable Web spider.

Background

According to whatis.com, a spider is "a program that visits Web sites and reads their pages and other information in order to create entries for a search engine index." A spider is only one piece of a Web search engine. We also need a tool to create a Web site search index (or collection) and a tool to search against the index with the user's search criterion. We'll be using a tool called Teleport Pro for the spider and Verity and ColdFusion to create and search against the collection.

You may be asking, "Why would I want to build a searchable Web spider?" The Verity 97 engine that's included with ColdFusion Server is designed to index content stored on your file system or in your database. The Verity engine's inability to build indexes by spidering your Web site creates a few problems, especially for complex, dynamically generated content sites. Significant differences can exist between raw content that's stored on your server and the way this content is presented to your customers. Another issue is the inability to include external links from your Web site in the search index.

There are many options for creating a searchable Web spider. Searchable Web spider tools can cost more than $30,000. There are also several application service providers that offer Web spider search engines at a significant cost. Either of these two expensive options is great for well-funded projects, but what are the rest of us supposed to use?

You may be thinking that you should just write your spider in C or, even worse, write it in ColdFusion using the CFHTTP tag. Why write something that probably already exists and besides, your valuable time should be used to improve your Web site. The real power of ColdFusion is its ability to glue together powerful widgets like Teleport and Verity to create a more powerful tool than either component.

Introducing the Spider, Teleport Pro

Teleport is designed to download Web sites to your hard drive for offline browsing, but we're going to use it to help create our searchable Web spider. When configured correctly, Teleport creates a mirror of Web sites to the file system while leaving paths, filenames, and URL parameters intact, for the most part. To create our searchable Web spider we're going to build a Verity collection from a locally stored mirror of a Web site. There are some issues with URLs containing characters that are not valid filenames, but we'll discuss them later.

Teleport can spider Web sites that use frames, image maps, HTML 4.0, CSS 2.0, DHTML, and XML. Teleport is multi-threaded and uses multiple virtual browsers to simultaneously download a Web site. There are different versions of Teleport, including one that's command-line driven, but we'll be using Teleport Pro, which costs about $40 for the full version and can be evaluated for free. Teleport Pro includes an easy to-use GUI and Web site download scheduler.

Setting Up the Spider

The first step in creating our searchable Web spider is to download and install Teleport Pro. Teleport Pro v1.29 supports Windows 95, 98, NT, and 2000, and is available in multiple languages. The evaluation version of Teleport can be downloaded at www.tenmax.com/teleport/pro/. Even though the evaluation version is limited to spidering 500 links, it'll be enough for this tutorial.

Now that Teleport is downloaded and installed on your development computer, we'll use it to create a mirror of the Web site we want to search against on the local hard drive. For this example, we're going to mirror HTML Goodies, a popular Web site for learning HTML.

From the Teleport Pro program, select File > New Project Wizard. Select the "Duplicate a Web site…" option and click the "Next" button. Enter "http://htmlgoodies.earthweb.com/" as the starting address. It can take a long time to download an entire Web site, so we'll limit the link search depth to two links from the starting point. Click on "Next." We only need text for our Verity collection, so select the "Just text" option and click "Next." Click "Finish" and save your Teleport project to "C:\SpiderData\HTMLGoodies.tpp".

To download all Web site file extensions and to prevent Teleport from adding .htm to the downloaded files, we must visit the Project Properties window. Select Project > Project Properties, then the "File Retrieval" tab. Select the

"Retrieve all files…" option and uncheck all the "Retrieval Modes" as seen in Figure 1. Now select the "Browsing/Mirroring" tab and uncheck everything except "Replicate the directory…" as in Figure 2. Click on "OK" and save the project again by selecting File > Save Project.

Now we're ready to download the Web site. If you have a low bandwidth connection to the Internet, spidering and downloading can take several minutes. If you're behind a proxy server, or want to use a connection other than your Windows default Internet connection, you'll have to visit the File > Proxy Server or File > Connection windows. When you're ready, start your project by selecting Project > Start.

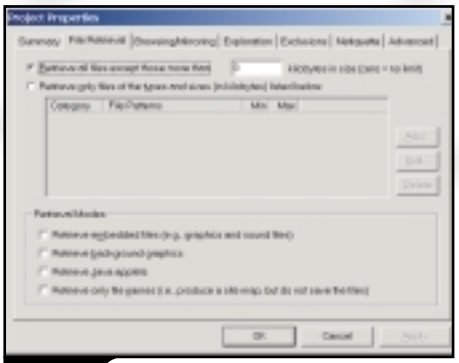


FIGURE 1: Teleport file retrieval options

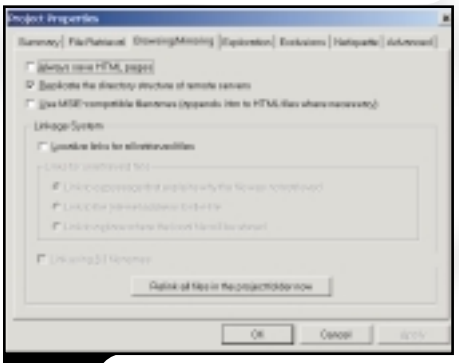


FIGURE 2: Teleport mirroring options

After the project has finished downloading the Web site, use Windows Explorer to browse the directory structure under "C:\SpiderData\". You'll discover the directory structure and files that Teleport Pro resolved by evaluating links from the HTML Goodies Web site.

Building the Verity Collection

Using the code in Listing 1, we can now create a new Verity collection that indexes the mirror of the HTML Goodies Web site that resides on the local file system. Our first task is to build the collection, which will act as a home to our Verity index.

```
<CFCOLLECTION
  ACTION="CREATE"
  COLLECTION="HTMLGoodies"
  PATH="C:\CFUSION\Verity\Collections\"
>
```

The ACTION="CREATE" parameter creates the directory structure for the index and various support files. The COLLECTION parameter specifies the name for the new collection. PATH indicates the location where the collection indexes and supporting files will be stored.

After defining the collection, we need to populate it with the CFINDEX tag as follows:

```
<CFINDEX COLLECTION="HTMLGoodies"
  ACTION="REFRESH"
  TYPE="PATH"
  KEY="C:\SpiderData\HTMLGoodies"
  RECURSE="YES"
  URLPATH="http://"
  EXTENSIONS=".htm,.html">
```

The COLLECTION parameter specifies the collection name that was created earlier by CFCOLLECTION. ACTION="REFRESH" tells Verity that we'll be adding to our existing collection, which at this point is empty. When the TYPE parameter is set to "PATH", the KEY parameter specifies the parent directory that the collection will be populated from. RECURSE="YES" tells Verity to include data from all the subdirectories below the parent directory specified in the KEY parameter.

The string in URLPATH is prepended to the CFSEARCH URL attribute. In our case this results in "http://" being concatenated with our CFSEARCH results such as "htmlgoodies.earthweb.com/index.htm".The EXTENSIONS parameter filters Verity indexing on a comma-separated list of file extensions so be sure to include all the possible extensions that may exist for the Web site that's being searched against. In our case, HTML Goodies consists mainly of static HTML files so .htm and .html filtering is appropriate. More detailed information about creating and optimizing Verity indexes is available from previous ***CFDJ*** articles.

Creating the Web Spider Search Templates

Now that we have a Verity collection populated with files mirrored from the HTML Goodies Web site, we need to create the templates to receive the user's search criterion, execute it against the HTML Goodies collection, and display the search results.

The template Search.cfm (see Listing 2) is a standard form that allows users to input a search criterion and decide if they want to see a search summary. Search.cfm posts to Search_Action.cfm where the real work occurs.

Search_Action.cfm executes the search criterion against the HTML Goodies Verity collection and displays the formatted search results as seen in Figure 3.

The code for Search_Action.cfm is in Listing 3 and unlike the Search.cfm form, some of the code deserves explanation.

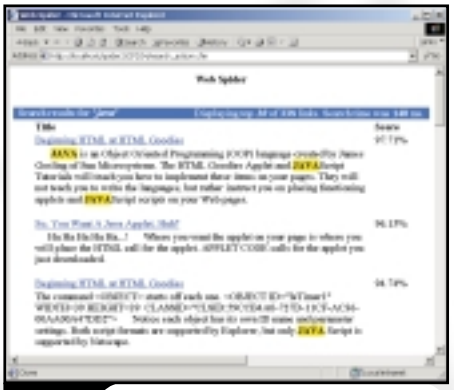


FIGURE 3: Mind map with one circuit app expanded

```
<CFSET TickBegin = GETTICKCOUNT()>
<CFSEARCH COLLECTION="HTMLGoodies"
          NAME="Search_Results"
          CRITERIA="#FORM.Search#"
          TYPE="SIMPLE">
<CFSET loopTime = GETTICKCOUNT() -
TickBegin>
```

The above CFSEARCH tag executes the search criterion entered by the user, #FORM.Search#, against the HTML Goodies collection and returns the resulting hits. It's important to note that CFSEARCH has a serious limitation – it throws an error if the search results exceed 64K in size. CFSEARCH will return a 64K limit error even if the CFSEARCH MAXROWS parameter is set to a low number. CFSEARCH may also throw an error if invalid search criterion is entered. There are various custom tags available to filter bad search criterion and TRY/CATCH blocks should be used around all CFSEARCH tags to trap error conditions.

Those of you who have done ColdFusion performance tuning will recognize the CFSETs around the CFSEARCH tag. GETTICKCOUNT () is a useful function because it returns the clock counter value in milliseconds (ms). In this case I'm using it to determine the execution time for CFSEARCH, but the same code can be used to optimize your code by timing sections of your CFML.

The next part of the code in Listing 3 simply displays some summary information about the CFSEARCH that just occurred. Search_Results.RECORDCOUNT is the total number of records returned from the CFSEARCH.

```
<CFOUTPUT QUERY="Search_Results"
          MAXROWS="#MaxLinks#">
```

The CFOUTPUT tag loops through the CFSEARCH results query and builds a new table row for each search hit. The MAXROWS parameter of CFOUTPUT limits the displayed results to 10 rows in this example because we've set MaxLinks = 10.

```
<CFSET Link= REPLACELIST
(Search_Results.URL,
"\index.htm", " ")>
```

There are quite a few interesting pieces to this seemingly simple REPLACELIST function so we'll take it one step at a time. Search_Results.URL is fundamentally the relative file path to the search hit under C:\SpiderData\HTMLGoodies with the back slashes reversed to forward slashes to make it a valid URL. Search_Results.URL is also the result of appending the slash-corrected string with the string value specified earlier by the URLPATH parameter in the CFINDEX tag (see Listing 1). In our case, Search_Results.URL equals the URLPATH http://" prepended to a string such as htmlgoodies.earthweb.com/beyond/intbrowtest.html". This results in the fully qualified URL "http://htmlgoodies.earthweb.com/beyond/intbrowtest.html".

When the Teleport Pro spider encounters a link such as http://htmlgoodies.earthweb.com/primers/, it can't determine the default template name so it assumes the name is index.htm. The actual URL could be http://htmlgoodies.earthweb.com/primers/default.cfm, but if the link to this page doesn't include the filename, Teleport Pro will mirror it to the local file system as htmlgoodies.earthweb.com\primers\index.htm. The REPLACELIST function replaces any "index.htm" strings with a space to eliminate this issue.

REPLACELIST allows the replacement of multiple substrings with different values. You might be asking yourself, since we're replacing only one string, why use REPLACELIST instead of the REPLACE function? It was done out of anticipation that you may want to replace "-" with "?" for Web sites that use URL parameters. Teleport Pro downloads a Web site to the local file system, and there are a few characters that are valid in a URL but are not legal as file system characters, including the question mark character used for URL parameter passing. Teleport Pro converts all illegal file system characters found in the URL to "-". Teleport Pro stores the URL http://www.semiconbay.com/go.cfm?CID=17987 as http://www.semiconbay.com/go.cfm CID=17987 to the file system because the question mark is not a valid file character. Modifying REPLACELIST to (Search_Results.URL, "\index.htm,-", " ,?") will put the question mark back into the URL, repairing the change made by Teleport Pro.

> **"The search criterion entered by the user may include logical operators and multiple search terms so the LISTFIRST function is used to find the first word of the search criterion"**

The following code displays the summary results for each search hit and highlights any keywords that were discovered.

```
<CFSET SearchWord =
LISTFIRST(FROM.Search,' ')>

<CFSET Context = REPLACENOCASE
(Search_Results.SUMMARY,
"#SearchWord#","<SPAN STYLE=
'background:FFFF00'>
<B>#UCASE(SearchWord)#</B>
</SPAN>",'ALL')>

#Context#<BR>
```

The search criterion entered by the user may include logical operators and multiple search terms so the LISTFIRST function is used to find the first word of the search criterion. If the user enters "browser AND test", SearchWord will equal "browser".

The next CFSET highlights the SearchWord if it exists in the summary results. Search_Results.Summary is a field that's returned by CFSEARCH. When CFINDEX is called, Verity "automagically" guesses which three sentences are most important to the document being indexed and stores up to 500 characters by default. The number of sentences stored and the maximum character size can be changed by editing "style.prm" in the file/style directory of any Verity collection. A good paper on this and other Verity details is available at www.allaire.com/handlers/index.cfm?ID=18429.

As seen from Figure 3, the summary results generated by Verity don't necessarily include the keyword we want to highlight. If we wanted to ensure that we showed the keyword context, CFFILE could be used to retrieve the HTML for each hit. We could then remove the HTML tags using REREPLACE, locate the first occurrence of the keyword, and then show the keyword highlighted in context. The first version of the Web spider used this method, but execution times increase by an order of magnitude. With search engines, speed is important so the trade-off was probably a good one.

Conclusion

Quite a few improvements can be made to our simple Web spider. By mapping a Web server virtual directory to C:\Spider-Data\, we could allow users to view cached versions of the Web pages similar to the way Google does. Scalability issues with Verity might be improved by distributing the load across multiple servers.

Adding Web sites to the Teleport spider can be done more powerfully by using the command-line driven version of Teleport or by replacing it with an even better offline browser. Downloading Web sites and updating and optimizing the Verity collections can be fully automated using the Teleport and/or the ColdFusion scheduler. This article should be a helpful starting point for building your own low-cost searchable Web spider. To view my implemented version, go to www.semiconbay.com, enter your search criterion into the "Search the Site" box in the top left, and click "go."

• • •

Editor's note: This article pertains to CF 4.5 or lower. Creating a Web spider using CF 5 will be covered in a future article.

About the Author

Michael Barr is the director of Internet technologies at the Silicon Valley company semiconbay (www.semiconbay.com). He's an Allaire Certified Professional and an active participant in the Bay area ColdFusion User Group (BACFUG).

mbarr@psiconsulting.com

Listing 1: Verity_Build.cfm – Build and populate the Verity collection

```
<HTML>
<HEAD>
   <TITLE>Verity Build</TITLE>
</HEAD>

<BODY>

<CFCOLLECTION
  ACTION="CREATE"
  COLLECTION="HTMLGoodies"
  PATH="C:\CFUSION\Verity\Collections\">

<CFINDEX COLLECTION="HTMLGoodies"
  ACTION="REFRESH"
  TYPE="PATH"
  KEY="C:\SpiderData\HTMLGoodies"
  RECURSE="YES"
  URLPATH="http://"
  EXTENSIONS=".htm,.html">

The collection has been created and populated.

</BODY>
</HTML>
```

Listing 2: Search.cfm – Search criterion form

```
<HTML>
<HEAD>
   <TITLE>Web Spider</TITLE>
</HEAD>
<BODY>

<DIV ALIGN="CENTER"><B>Web Spider</B></DIV><BR><BR>

<CFFORM ACTION="search_action.cfm" METHOD="POST">

Search Criterion:
<CFINPUT TYPE="TEXT"
         NAME="Search"
         REQUIRED="YES"
         MESSAGE="Enter a search criterion.">

<INPUT TYPE="SUBMIT" VALUE="Submit"><BR>

Show Context and Summary?
<CFINPUT TYPE="CHECKBOX"
         NAME="Context"
         CHECKED="YES">
</CFFORM>

</BODY>
</HTML>
```

Listing 3: Search_Action.cfm – Display search results

```
Variables:
MaxLinks         = Number of links displayed
Form.Search      = Search criterion
Form.Context     = Option button to show context/summary
Search_Results   = CFSEARCH results
   .URL          = File system path with slashes reversed
                   and appended to URLPATH from CFINDEX
   .Title        = Short title of the search result
   .Summary      = Automatically generated summary
   .Recordcount  = Number of records in the CFSEARCH
                   results
   .Score        = Decimal relevancy score
PScore           = Percentage relevancy score
SearchWord       = First word in search criterion
Context          = Summary with highlighted keyword
<HTML>
<HEAD>
--->
```

```
<TITLE>Web Spider</TITLE>
</HEAD>
<BODY>

<CFSET MaxLinks = 10>

<!--- Time CFSEARCH --->
<CFSET TickBegin = GETTICKCOUNT()>

<CFSEARCH COLLECTION="HTMLGoodies"
          NAME="Search_Results"
          CRITERIA="#FORM.Search#"
          TYPE="SIMPLE">
<CFSET loopTime = GETTICKCOUNT() - TickBegin>

<DIV ALIGN="CENTER"><B>Web Spider</B></DIV><BR><BR>

<!--- Search results titlebar --->
<TABLE BORDER="0"
       WIDTH="100%"
       CELLPADDING="0"
       CELLSPACING="0"
       BGCOLOR="3366cc">
<TR>
   <TD>
      <FONT COLOR="WHITE">  Search results for
      "<CFOUTPUT><B><I>#FORM.Search#</I></B></CFOUTPUT>"
      </FONT>
   </TD>
   <TD ALIGN="right">
      <FONT COLOR="WHITE"><CFOUTPUT>
      Displaying top <B><I>#MaxLinks#</I></B> of
      <B>#Search_Results.RECORDCOUNT#</B> links.
      Search time was <B>#LoopTime#</B> ms.  
      </CFOUTPUT></FONT>
   </TD>
</TR>
</TABLE>

<!--- Search results table --->
<TABLE BORDER="0" WIDTH="90%" ALIGN="CENTER">
<TR>
   <TD><B>Title</B></TD>
   <TD><B>Score</B></TD>
</TR>
<!--- Loop through search results --->
<CFOUTPUT QUERY="Search_Results" MAXROWS="#MaxLinks#">
<!--- Change file paths to valid http URLs --->
<CFSET Link=
REPLACELIST(Search_Results.URL, "/index.htm", " ")>
<CFSET PScore = Search_Results.SCORE *100>
<TR>
   <TD>
   <A HREF="#Link#">#Search_Results.TITLE#</A>
   </TD>
   <TD>#PScore#%</TD>
</TR>
<TR>
   <TD>
   <!--- If the context option button is clicked
         then show context/summary --->
   <CFIF ISDEFINED("FORM.Context")>
      <CFSET SearchWord = LISTFIRST(FORM.Search,' ')>
      <!--- Replace first word of search criterion
            with a highlighted version of itself --->
      <CFSET Context =
      REPLACENOCASE(Search_Results.SUMMARY,
      "#SearchWord#","<SPAN STYLE='background:FFFF00'>
      <B>#UCASE(SearchWord)#</B></SPAN>",'ALL')>
      #Context#<BR>
   </CFIF>
   <BR>
   </TD>
</TR>
</CFOUTPUT>
</TABLE>

</BODY>
</HTML>
```

CODE LISTING

The code listing for this article is also located at

www.sys-con.com/coldfusion/sourcec.cfm

A Cold Cup o'Joe – Java CFX Basics

Part 5 of 8

Leveraging write once, run anywhere strategy

BY GUY RISH

Prior to ColdFusion 4.5 a developer could only write CFX tags in a binary language that supported the dynamic linking capabilities of the target platform.

For Windows it was C/C++ and Delphi and the like, any language that could create a DLL. For the UNIX platforms, it was pretty much the same thing – only with shared objects. Naturally anytime a CFX was distributed it needed to be compiled for the different platforms or the source code was required – which is assuming, of course, that the CFX was even designed to be ported!

Now that CFX tags can be written in Java it is a write once and run anywhere strategy, which is already one of the major pillars of the Java philosophy. This makes things simple: unless native calls are made, the CFX is guaranteed to run on both platforms without change.

Keeping It Simple

The folks at Allaire did us (the ColdFusion developer community) a wonderful service by making the Java CFAPI fairly simple. In fact, creating a basic custom tag is no more difficult than implementing a Java interface.

```
public interface CustomTag
{
public void
processRequest(Request request,
Response response)
throws Exception;
}
```

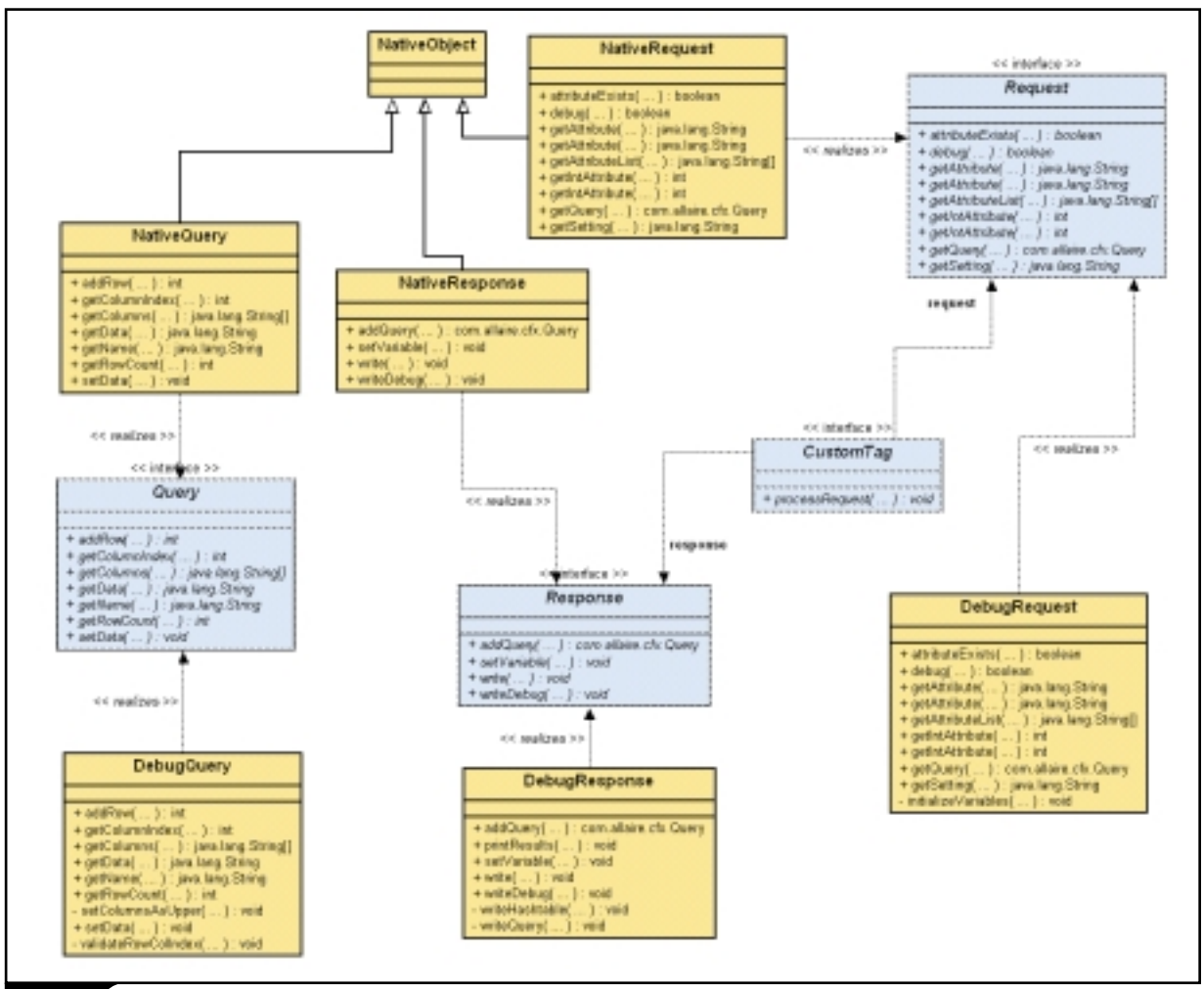


FIGURE 1: CFAPI UML model

The CustomTag interface in the CFAPI has only one method to implement: processRequest. This method receives two arguments, a Request object and a Response object, both from the CFAPI. It is through these two objects that the Java CFX receives information and provides output.

The CFAPI Model

All of the classes and interfaces of the CFAPI are shown in the UML model (see Figure 1).

There are a number of other classes and interfaces in the CFAPI library, but most of them are not designed for direct interaction with the developer. These are the native bindings and wrappers that hook into the ColdFusion Server. Of all of the classes in the CFAPI, there are only four you are likely to use directly: CustomTag, Request, Response, and Query.

CustomTag

This interface is the basis for all the custom tags derived from the CFAPI. Every tag must implement this interface in order for the ColdFusion Server to be able to hook it. There is only one method signature that needs to be implemented: processRequest (see Table 1). This method receives instances of two of the other classes: Request and Response.

Method	Description
processRequest	The CFXs entry

TABLE 1 CustomTag class

Request

This interface provides a way for the custom tag to receive information from the system or the calling logic (see Table 2).

Method	Description
attributeExists	Checks for the existence of a specific attribute.
debug	Checks for the presence of the Debug attribute.
getAttribute	An overloaded method, both versions retrieve the value (String) for the specified attribute, one sets a default value if the attribute doesn't exist.
getAttributeList	Returns a list of the received attributes.
getIntAttribute	An overloaded method, both versions retrieve the value (int) for the specified attribute, one sets a default value if the attribute doesn't exist.
getQuery	Retreives a passed query (the Query attribute).
getsetting	Retrieves a global setting from the CFX portion of the ColdFusion registry entries.

TABLE 2 Request class

Method	Description
addQuery	Adds a Query object to the values being passed back to the calling logic.
setVariable	Adds a variable to the values being passed back to the calling logic.
write	Writes a String directly to the output stream.
writeDebug	Writes a String directly to the debugging output stream.

TABLE 3 Response class

Method	Description
addRow	Adds an empty row to the set.
getColumnIndex	Gets the column index from the column name specified.
getColumns	Retrieves a list of the column names.
getData	Gets the value at the specified column and row.
getName	Gets the name of the Query instance.
getRowCount	Gets the number of rows in the Query.
setData	Sets the specified value for the specified column and row.

TABLE 4 Query class

Response

This interface provides a way for the custom tag to return information to the calling logic (see Table 3).

Query

This interface provides a way for the developer to manipulate a recordset (see Table 4).

Hello, World

I fear that I must exercise that tired example again; Listing 1 is a simple "Hello, World" Java CFX. As can be seen on Line 14, tag attributes can be retrieved through the Request object by means of the *getAttribute* method. As seen in Line 19, output can be returned to the ColdFusion Server by using the *write* method of the Response object.

Registering a Java CFX

As with their binary cousins, Java CFXs need to be registered in the ColdFusion Administrator. The process is fairly simple and – like almost all things in ColdFusion – does not require drilling through many screens.

The CFX Tags panel can be accessed from the ColdFusion Administrator. Starting from the left-hand navigation, under the Extensions menu is the hyperlink "CFX Tags". All the registered CFXs (binary and Java) are shown on the main CFX registration screen shown in Figure 2.

Enter a name for the CFX, set the type to be Java, and press the "Add" button. This will direct you to the second screen in the registration process, shown in Figure 3.

This screen is comprised of three entry fields: Tag Name, Class Name, and Description. The Tag Name field is the name by which the CFX will be referenced from within the template. In the case of the "Hello, World" example, created in Listing 1, I entered "Hello". So whenever the interpreter encounters a "cfx_hello" it will know to load the class file for this tag.

The Class Name entry specifies the class implementing the CFX's functionality. This class must exist somewhere in the CLASSPATH setting of the ColdFusion Server (configurable from the Java Setting panel, see Part 1 of this series in ***CFDJ***, Vol. 3, issue 1).

When necessary, be certain to use the full class name with package path. Typically I put my CFXs (Java or otherwise) in the CFX subdirectory of

FIGURE 2: Registered CFXs

the ColdFusion Server installation directory. This runs a little contrary to certain recommendations made in the ColdFusion documentation, which I will discuss later. The Description field is optional and can be used to describe the functionality of the library registered.

Testing the CFX

It is important to test the newly registered CFX since the ColdFusion Server does not verify any of the registration information until the CFX is first called. In the case of our "Hello, World" CFX this is rather simple. In Listing 2 I have created a simple template that called the "cfx_hello" tag in the body of an otherwise empty HTML document.

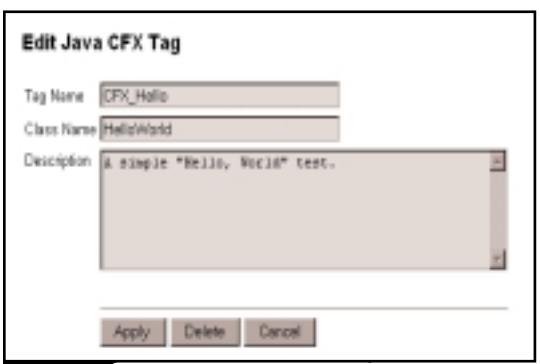


FIGURE 3: Registering a CFX

Hello, Query

Now that you have a grasp of the basics, let's step it up a bit. As a rule, CFXs cannot pass or return complex data types (the sole exception to this is the CFML query type). More specifically, the ColdFusion Server will allow for one CFML query to be passed into the CFX and one to be passed back. You can pass a CFML query in by specifying the name of the query in an attribute to the CFX named Query.

A reimplementation of the "Hello, World" CFX, shown in Listing 3, demonstrates how to work with an incoming CFML query. The template, shown in Listing 4, will query a database to retrieve a list of names, which it will pass to the CFX through the Query attribute.

The CFX is not that much of a departure from the "Hello, World" code in Listing 1. However, instead of receiving a single name in the attribute Name, the incoming query is looped over and the Name column is concatenated.

Reloading Problems

During development, code often gets changed rapidly and must be tested just as rapidly – and CFX tags are no exception. However, trying to rerun a Java CFX is not as simple as rerunning a newly compiled application. Once the CFX has been loaded by the JVM, it stays cached and the ColdFusion Server must be restarted to flush it from the JVM. The DebuggingFactory I created for Part 4 of this series (***CFDJ***, Vol.3, issue 6) is of little use here since the ColdFusion Server is invoking the CFX class and cannot be made to use that class.

There is actually a way to hijack Allaire's CFX class loader, but it's a sticky operation at best. The ColdFusion 4.5 documentation (Developing Web Applications with ColdFusion, Chapter 18: "Building Custom CFAPI Tags") states that if CFXs are deployed in the same configured directory as the cfx.jar file, the ColdFusion Server will check the timestamp on the class file and reload it if there have been changes. This can be further adjusted by passing an additional attribute to the CFX called Reload. For CFXs stored in the configured directory, the Reload attribute indicates to the ColdFusion Server whether the timestamp should be checked and what the reload rule should be (see Table 5).

This same documentation recommends that all Java CFX tags should be both debugged and deployed from this location. I would recommend against deploying production class files from the location for two reasons:

1. There is an additional performance hit to check the CFX class file each time for modifications.
2. It is a potential security risk to have a deployed production environment module reload without an explicit administrator request.

This situation gets a little better in ColdFusion 5.0, but the points I've made still remain.

CFX Limitations

For all their power (all CFXs – binary and Java alike), there are inherent limitations in the CFX model. These limitations have existed since the introduction of the CFX architecture and will continue to persist through version 5.0 (the Windows version being just newly available at the time of the writing of this article).

Singular Tags

CFX tags can only be singular tags. This means there's no end tag, and thus no body or subordinate tags. This effectively makes CFX tags analogous to simply executing external programs.

Value	Description
Auto	Check the timestamp on the file and reload if there's been a change.
Always	Always reload the class regardless of the timestamp on the file.
Never	Never reload the class file regardless of the timestamp on the file.

TABLE 5 Reload attribute value options

ABOUT THE AUTHOR
Guy Rish holds instructor certifications from Rational Software and Allaire and teaches Web development for SFSU MSP. He is active in both the ColdFusion and OO communities. He was the technical editor for Mastering ColdFusion 5 *from Sybex.*

Limited Relationship to Calling Logic

Undoubtedly CFX tags enhance the available power of the ColdFusion Server, but their implementation model hobbles them some. There is little connection with the calling logic of the CFX. There is no way for the CFX to find out information about its parent tag, it cannot trace the tag stack or access the Caller scope as other CFML-based tags do.

Complex Data Types

With the sole exception of a single incoming and outgoing query, no complex data types may be passed in or out of the CFX tag. This is a severe limitation that has not been corrected even in version 5.0 (there is a work around that I will discuss later in this series).

Difficulty in Debugging

Since the ColdFusion Server does not have specific hooks in it for stepping specific binary code or Java bytecode into either platform, debugging CFX tags can be difficult. Allaire has provided a number of classes to assist with this, but it isn't a silver bullet to the problem.

Wrapping It Up

Given the simplicity of writing CFXs in Java and the extensive libraries available, it makes sense to write tags in this manner. They are often more robust in functionality and frequently faster in execution than tags written in CFML. Naturally tags written in C/C++ will have better performance, but in many cases the loss of platform independence and thus the need to maintain two different code bases outweighs that gain.

In Part 6, I'll introduce you to some of the more advanced features of the CFX model, the recommended Allaire debugging techniques, and a couple of the benefits of using the JVM.

GUYRISH30@YAHOO.COM

Listing 1

```
 1.// import the Java CFAPI
 2.import com.allaire.cfx.* ;
 3.
 4.public class HelloWorld implements CustomTag
 5.{
 6.// custom tag entry point
 7.public void processRequest(Request request, Response
  response)
 8.throws Exception
 9.{
10.String strName = null;
11.String strGreeting = null;
12.
13.// retrieve/default the incoming argument
14.strName = request.getAttribute("NAME", "World");
15.
16.strGreeting = "Hello, " + strName;
17.
18.// write it back to the display stream
19.response.write(strGreeting);
20. }
21.}
```

Listing 2

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">

<html>
 <head>
  <title>Hello, World CFX</title>
 </head>

 <body>

  <cfx_hello>

 </body>
</html>
```

Listing 3

```
// import the Java CFAPI
import com.allaire.cfx.* ;

public class HelloQuery implements CustomTag
{
 // custom tag entry point
 public void
 processRequest(Request request, Response response)
 throws Exception
 {
  int nRowCount, nRowIndex, nColumnIndex;
  Query qry = null;

  if(request.attributeExists("QUERY"))
  {
   qry = request.getQuery();
  }
  else
  {
   throw new Exception("<h2>Error!</h2>No Query attribute
passed.");
  }

  nRowCount = qry.getRowCount();
  nColumnIndex = qry.getColumnIndex("NAME");

  for(nRowIndex = 1; nRowIndex <= nRowCount; nRowIndex++)
  {
   response.write("Hello, " + qry.getData(nRowIndex,
nColumnIndex) + "<br>");
  }
 }
}
```

Listing 4

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">

<html>
 <head>
  <title>Hello, Query CFX</title>
 </head>

 <body>

  <cfquery name="names" datasource="ccoj">
   select * from Person
  </cfquery>

  <cfx_helloquery query="names">

 </body>
</html>
```

CODE LISTING

The code listing for this article is also located at
www.sys-con.com/coldfusion/sourcec.cfm

Tiers, Not Tears

BY BEN FORTA

The basics of tiered development are very applicable to ColdFusion development

At last year's Developer's Conference I presented a session on creating *n*-tier (or multitier) applications in ColdFusion, explaining how tiered applications were more manageable and reusable. So why bring this up almost a year later? Well, a project I was working on recently forced me to revisit this topic, but this time for a whole new reason.

Sending Faxes with ColdFusion

A few years ago I wrote an application that needed to generate faxes programmatically. This particular application was used heavily, and generated thousands of faxes a day. To handle that kind of load I set up a dedicated fax-server – a computer equipped with multiple fax boards (each capable of sending multiple faxes simultaneously) and special software that turned data (text, graphics, and even application files) into faxes. The fax-server had multiple interfaces: a print driver that redirected output to the server, low-level APIs, a polling agent that picked up files from specified directories, and even an e-mail gateway.

Because I had no idea which interface would work best, I wrote a custom tag that black-boxed all the processing. I could simply pass data to a tag that looked like the following code, and all the heavy lifting could be tweaked and revised as needed:

```
<CF_SendFax TO_NAME=""
            TO_NUMBER=""
            FROM_NAME=""
            FROM_NUMBER=""
            SUBJECT="">
...
</CF_SendFax>
```

In the end I used the e-mail interface, which provided the simplest integration from a CFML coding perspective. I was able to use <CFMAIL> to send e-mail to a special mailbox (the TO attribute contained the recipient fax number, the message body contained the cover sheet, and so on). Clean and simple.

A month or so ago I needed to send faxes programmatically again, only this time for a low-volume application running on a Windows 2000 server. So, instead of setting up an expensive high-end fax server, I opted to use the fax services built right into Windows 2000 (yes, Windows 2000 installs a true fax service if you have a fax modem installed). The interface to this service is COM, so using <CFOBJECT> and a series of supporting <CFSET> tags (and some help from Dain Anderson of www.cfcomet.com fame), I ended up with a custom tag that could send faxes via this service. The interface was completely different: COM versus e-mail, files containing content versus message bodies, passing values as properties versus mail headers, and Windows-only versus multiplatform. What did my new custom tag look like? I'll show you:

```
<CF_SendFax TO_NAME=""
            TO_NUMBER=""
            FROM_NAME=""
            FROM_NUMBER=""
            SUBJECT="">
...
</CF_SendFax>
```

Note: If you'd like to know more about the Windows 2000 fax service and Dain's CF+COM integration code, see the "Extending ColdFusion with COM" chapter in my new *Advanced ColdFusion 5 Application Development* (Sept. 2001, QUE).

It's All About Encapsulation

So, two very different faxing solutions with very different capabilities and very different interfaces. Yet from within my CF code they're essentially interchangeable.

What makes this all work is encapsulation: taking a process and black-boxing it within a clearly defined interface so the inner workings are hidden and isolated. Encapsulation in ColdFusion is achieved via custom tags in much the same way as Java developers write beans and Windows developers write DLLs. And just like writing beans or DLLs, encapsulating processes within custom tags requires careful planning. A well-designed encapsulation as a custom tag must…

- Provide a clean and consistent interface to simplify invocation
- Not require extensive understanding of the internal workings
- Not have dependencies on external objects, code, or assumptions – code should just work as is
- Be well documented

Of course, custom tags are the primary way developers reuse code in ColdFusion (something I've written about extensively in this column), but there's more to it than reuse. Encapsulation also helps address two other important issues: group development and portability.

Group Development

Once upon a time (which was not that long ago in human years), all development was done by individuals – often someone with no social life, lots of caffeine, and a stack of books on everything from IP to JavaScript to creating animated GIFs. Fortunately, we've gotten past that

stage; development is (or should be) a well-managed process, complete with schedules, project management, and groups of developers with complementary areas of expertise.

That's a good thing, except that all the little bits that make up ColdFusion development (and indeed all Web development) tend to be so interrelated that developers need to know lots of things about lots of technologies. This of course makes it hard to find developers, harder to keep them, and even harder to train new ones.

Which is why tiered code and encapsulation are so important. Suppose your GUI gurus create a sophisticated menuing and navigational system, or your DBA redesigns your database schemas, or your resident COM expert implements the aforementioned fax services integration. If all these were encapsulated properly, any developer could use the underlying technologies without special training, without needing lots of help, and without making all sorts of horrible mistakes. And if (or when) those same gurus and experts changed or updated their creations, the developers using them wouldn't have to restart from scratch.

Tiering your code – encapsulating functionality or processes or components – is the key to successful group development.

Portability

Portability refers to the ability of code to run on different systems without requiring changes. For the most part, portability is of no real concern to ColdFusion developers. After all, I don't know many ColdFusion installations that suddenly needed to move from Linux to Solaris, or from Windows to HP/UX.

So why is portability of interest? Back to the fax example I started off with. Portability is impacted not only by the underlying operating system, but also by any external services – like faxing. The fax service I just used is very operating system–specific. It runs on Windows 2000 only (not even on Windows NT or Windows ME). Does that mean I shouldn't use it? Absolutely not. There's nothing wrong with using whatever services and features that are available to you. If it's there, use it.

At the same time, you must protect yourself and make sure you aren't marrying your code to external processes you can't control. In my case, if the fax service goes away in the future, or if I have to run the app on Linux, or if I suddenly find that the volume of faxes has exceeded the abilities of this simple service…well, then, I'll find another faxing solution and create a tag with the same name and attributes to encapsulate the calls. None of my code will break; any needed changes will be local and controlled; and, for now, I can go on using what works because, well, it works for me right now.

In other words, why worry about portability until you really have to? For now, just make sure any code that could present portability problems is appropriately encapsulated. Worry about portability if that ever actually becomes necessary.

For the record, this isn't a new concept. Internally, this is exactly how ColdFusion works. <CFFTP> is implemented on Windows in a very different way than it is on UNIX, but CF developers don't need to know this. The underlying code is different, but the functionality and interfaces are the same, thanks to encapsulation.

Summary

Tiered development is nothing new, and *n*-tier development theory is the kind of stuff that makes stuffy academics break out in smiles. True *n*-tier development can be a pain to implement, but the basics of tiered development – breaking code into logical chunks with the appropriate levels of encapsulation – are very applicable to ColdFusion development. Keep this in mind the next time you design an application. The extra effort is minimal, and the benefits are staggering.

ABOUT THE AUTHOR

Ben Forta is Macromedia's senior product evangelist and the author of numerous books including the recently published ColdFusion 5 Web Application Construction Kit *and its sequel,* Advanced ColdFusion 5 Development. *For more information on Ben's books visit www.forta.com.*

BEN@FORTA.COM

SYS-CON MEDIA

135 Chestnut Ridge Rd., Montvale, NJ 07645
Telephone: 201 802-3000 **Fax:** 201 782-9600

president and ceo
Fuat Kircaali fuat@sys-con.com

advertising
senior vp, sales & marketing
Carmen Gonzalez carmen@sys-con.com
vp, sales & marketing
Miles Silverman miles@sys-con.com
advertising director
Robyn Forma robyn@sys-con.com
advertising account manager
Megan Ring megan@sys-con.com
associate sales manager
Carrie Gebert carrie@sys-con.com
associate sales manager
Alisa Catalano alisa@sys-con.com
advertising intern
Alison Novick alison@sys-con.com

editorial
executive editor
M'lou Pinkham mpinkham@sys-con.com
editor
Nancy Valentine nancy@sys-con.com
managing editor
Cheryl Van Sise cheryl@sys-con.com
associate editor
Jamie Matusow jamie@sys-con.com
associate editor
Gail Schultz gail@sys-con.com
associate editor
Brenda Greene brenda@sys-con.com
assistant editor
Lin Goetz lin@sys-con.com

production
vice president, production
Jim Morgan jim@sys-con.com
art director
Alex Botero alex@sys-con.com
assistant art director
Cathryn Burak cathyb@sys-con.com
assistant art director
Louis F. Cuffari louis@sys-con.com
assistant art director
Richard Silverberg richards@sys-con.com
graphic designer
Abraham Addo abraham@sys-con.com
graphic designer
Aarathi Venkataraman aartathi@sys-con.com

web services
web designer
Stephen Kilmurray stephen@sys-con.com
web designer
Purva Dave purva@sys-con.com
web design intern
Carol Auslander carol@sys-con.com

accounting
chief financial officer
Bruce Kanner bruce@sys-con.com
assistant controller
Judith Calnan judith@sys-con.com
accounts payable
Joan Larose joan@sys-con.com

sys-con events
vice president, events
Cathy Walters cathyw@sys-con.com
sales executive, exhibits
Richard Anderson richard@sys-con.com
sales executive, exhibits
Michael Pesnick michael@sys-con.com
conference director
Danielle Nappi danielle@sys-con.com
conference manager
Michael Lynch mike@sys-con.com
show assistant
Niki Panagopoulos niki@sys-con.com
JDJ store manager
Anthony D. Spitzer tony@sys-con.com

Ask the **Training Staff**

A source for your CF-related questions

BY
BRUCE VAN HORN

I hope that all of you are enjoying your summer and getting in plenty of R&R. If you aren't, I hope you have at least upgraded to the new CF Server 5.0.

Not that this is better than a week at the beach, but it should certainly bring a little joy into your life because of some of the new features! Anyway, upgrade if you haven't already, start playing with the new features, and keep those questions coming. Here are some I've answered recently.

Q: *How can I use CF to dynamically generate (output) special ASCII characters, such as the tab or line break characters?*

A: The easiest way to do this is by using the Chr() function. Simply plug in the ASCII value of the character you want generated. For example, for a tab character, use #chr(09)# in your code. If you're not sure what the ASCII code is for the character you want, use this simple code to loop through all the values:

```
<cfoutput>
<cfloop from="1" to="256"
index="i">
ASCII Code #i#: #chr(i)#<br>
</cfloop>
</cfoutput>
```

Q: *You said in your class to avoid using a text file as a data source, but I have a situation where I need to do that. How do I use a text file as a data source in CF?*

A: Even though it's a bad situation, it's a good question. The first thing you need to understand is how to create the data source. When creating the data source in the CF Administrator, avoid the temptation to point it to the actual file. Create your data source so it's pointing only to the subdirectory that the file is in. If you point it to the actual file, the connection will fail.

Once the data source is created in CF, you can query the file using the CFQUERY tag as you would for any other database. The biggest difference is in the FROM clause in your SQL. Here's where you'll use the name of the file. For example:

```
<CFQUERY NAME="MyQuery" DATA-
SOURCE="TextTest">
 SELECT Column1, Column2, etc...
 FROM   MyTextData.txt
</CFQUERY>
```

Q: *In CF Studio, when I go to click on one of the resource tabs (like the database tab), it frequently unlocks the tab from the rest of the resource area, which is really annoying. Is there a way to stop this?*

A: I know exactly what you mean and I agree that it's very annoying! If you're using Studio 4.5.2 there's a way to stop this "feature." Go to the Options menu and choose Settings. In the lower-right corner of the main screen there's an option called "Lock Tabs". If you check this box, your resource tabs will stay where they are and your life will be less stressful!

Q: *I've upgraded to CF Server 5.0 and am trying to view the examples for the new CFGRAPH tag, but nothing ever shows up. What am I doing wrong?*

A: The CFGRAPH tag is a cool addition to ColdFusion, but it doesn't work unless you start a separate service called the *ColdFusion Graphing Server*. By default this service is not started automatically. If you're running Windows NT or 2000, go to the Service Manager, start the service, and then try the examples. They should work now.

Q: *What's the best way to clear out a cached query?*

A: It depends on how you cached it. If you used the CACHEDWITHIN attribute of the CFQUERY tag, you can run the same query with CACHEDWITHIN="#CreateTimeSpan(0,0,0,0)#". This will refresh the query. If you cached the query using a persistent variable scope, such as session or application, you can use the StructDelete() function to remove the query from the scope. For example, if you cached the query using something like <cfquery name="Application.MyQuery"...>, you can delete it using this statement:

```
<cfset tmp = StructDelete-
(Application,"MyQuery")>
```

• • •

Please send your questions about ColdFusion (CFML, CF Server, or CF Studio) to AskCFDJ@sys-con.com. Please visit our archive site at www.NetsiteDynamics.com/AskCFDJ.

ABOUT THE AUTHOR

Bruce Van Horn is president of Netsite Dynamics, LLC, a certified ColdFusion developer/instructor, and a member of the **CFDJ** *International Advisory Board.*

BRUCE@NETSITEDYNAMICS.COM

Essential Technologies for Today's B2B Integration Challenges

Web Services EDGE

PRE-REGISTER ONLINE TODAY
SAVE $300

SEPTEMBER

New York, NY

web services conference&expo EAST COAST — 24-25

JDJ EDGE conference&expo — 23-26

OCTOBER

Santa Clara, CA

web services conference&expo WEST COAST — 24-25

XML EDGE conference&expo 2001 — 22-25

Participating Companies, Sponsors and Exhibitors as of June 1, 2001

TogetherSoft, SilverStream, sitraka software, sonic software, aligo, KENAMEA, shinka, SAVAJE, INFERDATA, bea, InCert, SEAGULL, Data Quad, idoox, CAPE CLEAR, ParaSoft, MERANT, PointBase, hp invent, pingtel, TOPCODER, NOVA LABS, Bestning, Rational, ROGUE WAVE SOFTWARE, OASIS, ORACLE, DataMirror, CodeHound.com, DATA JUNCTION, ENTEGRITY Solutions, TOWER, xml spy, TALKING BLOCKS, WHITEHILL, AdvanTel, XML.ORG, XMLCities, XML Journal, Programmer's Paradise, ColdFusion Developer's Journal, WebSphere, WebServices, WebLogic, webtechniques, JAVA Developer's Journal, SD Times, wireless

each venue features:

- Over 100 Information-Packed Sessions
- Two Distinct Conferences for Just One Registration Fee
- Pioneers and Visionaries
- Leading Vendors Presenting New and Problem-Solving Products

ColdFusion fasttrack
SEPT. 24-25, 2001
TWO
CONCENTRATED DAYS OF LEARNING TO JUMPSTART YOUR PROFESSIONAL CREDENTIALS!

Secure Your Seat Now!

Two Easy Ways to Register

1 On the Web:
www.SYS-CON.com

2 By Telephone:
201 802-3069

Be sure to visit www.SYS-CON.com for travel information.
Reserve Your Hotel Room Early!

<dot.com>
- buyer's guide
- coldfusion forums
- mailing list
- coldfusion jobs
- coldfusion store

<magazine>
- advertise
- authors
- customer service
- editorial board
- subscribe

<content>
- archives
- digital edition
- editorial
- features
- interviews
- product reviews

<conferences>
- coldfusion edge fast track new york, ny sept 23–26

<search cfdj>
Search

<cfdj specials>
Collect All Three!
JDJStore.com $175.99
JDJ, XML, CFDJ - ALL THREE!!!-The Complete Works

What's Online

www.sys-con.com/coldfusion

CFDJ Online

Check in everyday for up-to-the-minute news, events, and developments in the ColdFusion industry. Visit www.sys-con.com/coldfusion and be the first to know what's happening.

ColdFusion Edge 2001 Conference Schedule Available

Go to www.sys-con.com/coldfusionedge to download the conference brochure for the ColdFusion Track at **JavaEdge 2001,** taking place at the New York Hilton, New York City, September 23–26, 2001. The conference offers ColdFusion developers, both beginner and advanced, comprehensive technical sessions designed to prepare them for the ColdFusion 4.5 Certified Developer Exam.

Registration, keynotes, sponsors, travel information, and a list of exhibitors and sponsors are also available.

ColdFusion Forum

Join ColdFusion List, the new ColdFusion community. Join other IT professionals, industry gurus, and ColdFusion writers for ColdFusion discussions, technical questions, and more.

Voice your opinions and assessments on topical issues or hear what others have to say.

Join the ColdFusion List now to monitor the pulse of the ColdFusion industry.

JDJ Store CD Special

The complete library of ***CFDJ***, ***JDJ***, and ***XML-J***, articles are available on CD at a special price for a limited time.

Order now and have more than 1,000 articles on hand for research and review. There are features, how-tos, product reviews, tips and tricks, interviews, IMHOs, and more!

CFDJ topics include Tips from Ben Forta, Custom Tags, ColdFusion and Java, Server Stability, Site Performance, Using XML and XSLT with ColdFusion, Fusebox, Building E-Business Apps, Application Frameworks, Error Handling, Wireless ColdFusion, Ask the Training Staff, Authentication, Monitoring, Smart Objects, A Beginner's Guide to CF, Safe Scripting, JavaScript, and Load-Balanced Servers.

This exclusive package normally sells for $260, but it can now be yours for only $175.99.Order today and save!

Ben Forta's ColdFusion Tip of the Day

Click here for ColdFusion tips, links, tags, and resources from Allaire's CF evangelist. A new tip every day!

Subscribe to Our Free Weekly Newsletters

Now you can have the latest industry news delivered to you every week. **SYS-CON** newsletters are the easiest way to keep ahead of the pack. Register for your free newsletter today! There's one for ColdFusion, Java, XML, Web Services, and Wireless. Choose one or choose them all!

Array Launches HotQuery 1.9

Array Software Inc. has released HotQuery version 1.9, the first 100% browser-based database query-by-example engine and reporting tool for Oracle, MS SQL Server, and Sybase. Users can query tables and views through the HotQuery-by-example point-and-click interface or through a SQL query writer. Queries can be saved and run anytime, anywhere.

New features include HotPrompts for dynamically entering criteria at query runtime; HotSort, a point-and-click interface for sorting query results; and HotJoin, an intuitive, wizard-driven interface for creating multi-table joins. A full-featured Web-based demo is available at www.hotquery.com.

DCSE Releases Map Library Portal

(*Mission Viejo, CA*) – DCSE Inc., a California developer of collaborative Web-based project solutions, announced a new product line, Map Library Portal. Designed to increase end-user access to an increasing number of GIS maps developed by the growing ESRI user community, the application is built with Macromedia ColdFusion and SQL Server.

Map Library Portal is a Web-based collaboration server that enables users to find maps for a specific purpose at a given time. It provides a scalable application for project participants to share and connect project-critical maps.

www.dcse.com

SYS-CON *Named Winner of the 2001 New Jersey Technology Fast 50 Award*

(*Montvale, NJ*) – **SYS-CON Media** (www.sys-con.com), the world's leading *i*-technology publisher and the producer of *i*-technology developer conferences, has been named by Deloitte & Touche to its annual list of the 50 fastest-growing technology companies in New Jersey. As a winner of this award, **SYS-CON** was also nominated for the national competition for the 500 fastest-growing technology companies in the U.S. – the Technology Fast 500 Award.

In 1999, **SYS-CON Media** was ranked 194th by *Inc. 500* on the list of America's fastest-growing privately owned companies and was nominated again in 2001.

"We are delighted to be a 2001 New Jersey Technology Fast 50 winner," said Fuat Kircaali, founder and CEO of **SYS-CON Media**. "Although we've experienced consistent growth since 1994, the year the company was founded, only after **SYS-CON** turned 5-years-old were we eligible for these nominations. This year we are confident that we will be named as one of America's fastest-growing companies once again by *Inc. 500*, as well as by the Technology Fast 500."

SYS-CON received the Technology Fast 50 Award based on a 1,075% 5-year revenue growth between 1996 and 2000, and the 1999 *Inc. 500* award based on a 1,307% 5-year revenue growth between 1994 and 1998.

ADVERTISER INDEX