ANNOUNCING: **WEB SERVICES EDGE CONFERENCE & EXPO** page 23

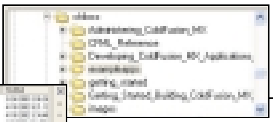
U.S. $8.99 (Canada $9.99)

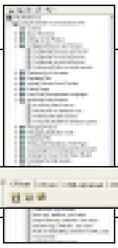
ColdFusion Developer's Journal

ColdFusionJournal.com

June 2003 Volume: 5 Issue: 6

Announcing...
2003 CFDJ Readers' Choice Awards
2003 CFDJ Readers' Choice Awards
VOTE NOW!
GO TO WWW.SYS-CON.COM





RETAILERS PLEASE DISPLAY UNTIL AUGUST 31, 2003

$8.99US $9.99CAN

SYS-CON MEDIA

FUSETALK

www.fusetalk.com/new

EDGE WEB HOSTING

www.edgewebhosting.net

ACTIVEPDF

www.cfconf.org/cfun-03

editorial offices

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

Let Your CF Voice Be Heard

Another month in the world of CF has passed, and as always, it's certainly not been a dull one. This month's editorial will be devoted to a handful of important ***CFDJ*** housekeeping items before you turn the page to get into the technical meat. The 3rd Annual *CFDJ* Readers' Choice Awards are now in full swing with hundreds of votes in so far and more coming in even as I write.

By Robert Diamond

I'm also proud to note that we've received lots of positive feedback from around the industry for the new fraud protection and voting systems that we've added to this year's awards program. The most notable addition is that current subscribers' votes count as 2 points, giving them more weight in the overall standings than new ***CFDJ*** visitors' votes, which count as 1 point. It's been interesting to observe the breakdown of these numbers as it's a good reflection of who we're reaching. Voting continues until the end of August, so while the deadline is still a while off, if you haven't voted yet, come cast your vote and let your voice be heard (www.sys-con.com/coldfusion/readerschoice2003).

Speaking of letting your voice be heard, we're going to be doing some "Quick Polls" on the *CFDJ* Web site over the next few months, looking for your opinion on the industry, *CFDJ*'s content (both print and online), and several other issues of the day that will help guide us in our editorial coverage, as well as our mission to constantly improve and to serve you better. As always, feel free to drop me a line anytime at robert@sys-con.com with questions, comments, complaints, and of course, brilliant ideas.

Here's yet another way to voice your opinion in the *CFDJ* world. We'll be doing a *CFDJ* Keynote Panel discussion at the CFUN 2003 Conference, June 21–22, 2003, in Gathersburg, Maryland (www.cfconf.org/cfun-03). Appearing on the panel will be many of the authors that you see here each month in the magazine, and moderating will be yours truly. We'll be covering all the latest issues in the ColdFusion world – including technical questions, where CF is going, and lots of other issues of the day.

We'll be sticking to a format similar to last year's CFNorth Conference, so you can submit your questions to me in advance via e-mail if you'll be attending the conference. When we run out of pre-submitted questions, we'll open it up to the floor. Also similar to last year's event, we'll cover the highlights in a subsequent issue of *CFDJ*. I've always been a big advocate of conference education, and if you can make it – I'll see you there.

One last bit of *CFDJ* news is that Charlie Arehart, a frequent writer for *CFDJ*, and one of its co-technical editors, recently accepted the position of chief technical officer for New Atlanta Communications, most known in the CF world as the makers of BlueDragon. His role with the magazine, and indeed within the community as an evangelist for CFML, will not change. As this month's and future articles will show, he's still got plenty of topics to talk about that are of interest to all CF and CFML developers. We're looking forward to reading them!

About the Author

Robert Diamond is vice president of information systems for SYS-CON Media, and editor-in-chief of **ColdFusion Developer's Journal.** *Named one of the "Top thirty magazine industry executives under the age of 30" in* Folio *magazine's November 2000 issue, Robert holds a BS degree in information management and technology from the School of Information Studies at Syracuse University. Visit his blog at* www.robertdiamond.com.

robert@sys-con.com

NEW ATLANTA COMMUNICATIONS

www.newatlanta.com

SYS-CON MEDIA

president & ceo
Fuat A. Kircaali fuat@sys-con.com

business development
vp, business development
Grisha Davida grisha@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
director of sales & marketing
Megan Mussa megan@sys-con.com
advertising sales manager
Alisa Catalano alisa@sys-con.com
associate sales managers
Carrie Gebert carrieg@sys-con.com
Kristin Kuhnle kristin@sys-con.com

sys-con events
president, events
Grisha Davida grisha@sys-con.com
conference manager
Michael Lynch mike@sys-con.com

customer relations
circulation service coordinators
Niki Panagopoulos niki@sys-con.com
Shelia Dickerson shelia@sys-con.com
manager, jdj store
Rachel McGouran rachel@sys-con.com

sys-con.com
vp, information systems
Robert Diamond robert@sys-con.com

web designers
Stephen Kilmurray stephen@sys-con.com
Christopher Croce chris@sys-con.com

online editor
Lin Goetz lin@sys-con.com

accounting
financial analyst
Joan LaRose joan@sys-con.com
accounts receivable
Kerri Von Achen kerri@sys-con.com
accounts payable
Betty White betty@sys-con.com

subscriptions
Subscribe@sys-con.com
1 888 303-5282
For subscriptions and requests for bulk orders, please send your letters to Subscription Department
Cover Price: $8.99/issue
Domestic: $89.99/yr (12 issues)
Canada/Mexico: $99.99/yr
all other countries $129.99/yr
(U.S. Banks or Money Orders)
Back issues: $12 U.S. $15 all others

Tales from the List

Database data extraction – why loop? In fact, why do anything in CFML at all?

This month we're returning to our regular format of showcasing a thread from the CFDJList mail-list server. The thread we will review reminded me that there isn't always one right answer – often there are many alternatives to one approach. Each comes with its own advantages and disadvantages, as well as performance benefits.

By Simon Horwith

The thread began with a post by Chris Edwards of Outer Banks Internet (www.outerbanksinternet.com). Chris wrote to the list because he had written a chunk of code to extract a recordset and loop over that recordset in order to write a part of each record to a text file, and that code was performing unsatisfactorily.

In a nutshell, Chris informed the List that he was running a loop over a query that ran in excess of 22,000 iterations as he looped. He was setting an "e-mail" variable equal to itself plus the e-mail from the current iteration of the loop. After the loop, he was writing the variable (which now contained a list of e-mail addresses) to a file on the server using <cffile>. Chris found that running this code used roughly 800+ MB of RAM.

Many List participants replied, each offering a suggestion that would improve performance. Ray Thompson, a regular List participant, immediately suggested writing each e-mail address to file as he looped, rather than storing them all in memory before writing them to list. I immediately replied, suggesting that Chris should avoid looping so many times and that he might want to break the loop into several smaller loops and then concatenate those results.

I then responded a second time with what I thought would be the absolute best solution for Chris: don't loop at all! After reading his post a second time and reviewing his existing code, it occurred to me that what he was doing was looping over a massive query in order to append the value of one column after each loop. I realized that rather than looping, Chris could use the #valueList()# function to grab the list of e-mail addresses rather than loop over them in order to build a string containing the same data.

Andre Turrettini responded with the suggestion that Chris might want to loop over the query and store the values in an array rather than a string, because, in his experience, string manipulation is slower in most environments. Not a bad suggestion at all. Complex data structures are designed for just this sort of thing.

Then Douglas Knudsen chimed in and put all of our solutions to shame. He suggested making the database do the work instead, which makes perfect sense in retrospect. He then went into more detail by specifying that what needed to be done was a combination of my suggestion of using the #valueList()# function and trying a SQL snippet similar to his pseudocode (but one specific to Chris's MySQL database platform): "`SELECT email || chr(13) || chr(10) FROM Foo....`" and then simply writing the result to file. The SQL code Douglas posted essentially retrieves all e-mail addresses, each appended with new line characters. Chris' solution was found!

All suggestions made were valid, and I was correct in stating that there was no need to

—continued on page 45

About the Author

Simon Horwith has been using ColdFusion since version 1.5. He is a Macromedia Certified Advanced ColdFusion and Flash developer and is a Macromedia Certified instructor. In addition to administering the CFDJList List Serve and presenting at CFUGs and conferences around the world, he has also been a contributing author of several books and technical papers. Originally from the Washington, DC area, Simon is currently working as a private consultant in London, England.

simon@horwith.com

System Thinking

Welcome to the majors

"It is not enough to do your best; you must know what to do, and then do your best."

– W. Edwards Deming

By Hal Helms

In several recent ***CFDJ*** articles, I've described software architecture as akin to model building. In both designing software models and building scale models, it's important that the model be internally consistent as well as sufficiently rich to encompass the desired behaviors of the real-world system.

For example, if you want to create a model car, you will want to ensure that it's internally consistent (turning the wheels does not cause the hood to open, for example) and rich enough so that if, for another example, we open the hood of the car, we see a model engine.

Software, of course, isn't nearly so visual as a scale model – a fact that makes the discipline of software architecture difficult: when you set about building a software version of a model car, all the construction occurs in your mind. The problem is that brains aren't particularly good at storing large amounts of complex information, so our species has developed writing and drawing as ways of documenting our thoughts. Software architects employ a variety of tools to translate the invisible to the visible – from cocktail napkins to UML diagramming programs.

But static scale models themselves are not capable of doing the real work needed to provide benefit to users. A superficial object model analysis may determine that we need (for example) both a Car and an Engine object, but this alone doesn't specify how the two will work together.

Identifying objects isn't enough; we must make sure that those objects interact properly. And that's precisely the definition of a system: a group of interacting, interrelated, or interdependent elements forming a complex whole.

In my simple example of a Car and an Engine, it's simple enough to see their relationship, but the more complex an application is, the more it must do, or the more likely it is that our application will change over time, the more important it is that we build a system that is flexible enough to withstand the shocks of future changes – that we have thought through the best ways for our objects to interact.

Too often though, a software application is seen as a solution to a specific problem. We (hopefully) find out what users want and then build it for them. But this approach misses a key ingredient of virtually all software requirements – that the requirements themselves will change as the business environment changes.

In more stable times, software could be built and counted on to run without alteration for years, even decades. But the pace of change has accelerated to a point where software is often out of date before it is even released (and this is a large cause for the high degree of software failure). Therefore the need for good systems analysis is greater now than ever.

But how shall we design systems for future environments, which, by their very nature, are unknown? Over many years of building software, we have found that the most successful software (in terms of longevity) has certain common features, such as information hiding, encapsulation, and loose coupling. A great deal of theoretical understanding has been gained from the study of both successful and failed software systems.

Does such theoretical knowledge help those of us in the trenches, though? It does – both in the design of modern languages such as Java and, of more immediate use, in the rise of concrete guidance on how to design flexible systems of object interactions. Such guidance is known by the term, *design patterns.*

Design patterns are unique in that they offer help on deciding how objects should interact – how, in essence, to create mini-systems that are maintainable. If that sounds too abstract to you, consider a design pattern in a field different from our own. (Apologies to our international readers who may not be that familiar with the American pastime, baseball.)

A Practical Example

By a wave of my magic wand (handily kept for just such occasions), you have been turned into the shortstop for your favorite baseball team. (If you've ever read the wonderful book, *The Year the Yankees Lost the Pennant*, or seen the movie based on it, "Damn Yankees," you'll recognize the situation: a regular guy suddenly becomes a major league player. In the book, there was some soul-dealing involved with the devil;

COOLFUSION

www.coolfusion.com

luckily your situation has no such dangers.)

You've got the fans, the salary, and (most profitably) the endorsements. But alas, you lack experience. There are limits, it seems, to even the most magical of wands and some things cannot be granted; they must be learned.

As you take the field, you try to recall the games you've seen on television, hoping for some guidance. You search for memories of other shortstops you've seen, trying to remember whether the shortstop stands between first and second base or second and third base. Since the second baseman is already somewhere between first and second base, you casually jog to the empty spot between second and third. When no one laughs, you breathe a sigh of relief: one hurdle passed.

You find yourself hoping that the opposing batters will hit long fly balls that will be fielded by outfielders. The first batter is up and does exactly what you were hoping – almost: on the first pitch, he hits a long fly ball – so long, in fact, that it easily clears the right field wall for a homerun.

The second batter comes up to the plate, and the pitcher, shaken by such an early homerun, walks him. With a man on base, the infield (which you're part of) becomes more attentive; the second and third basemen shift their positions. The next batter takes two strikes then pops up a very short fly ball. You watch the arc of the lazily hit ball until you realize that it's coming down very close to you. You think you can catch it, but should you then throw it to first base? To second base? Back to the pitcher?

The ball descends but before it arrives, the umpire calls: "You're out!" and the batter begins to trot out to the dugout. But you haven't caught the ball, yet! When it finally lands in your glove, the pitcher is already standing with his glove toward you, waiting for you to throw it back.

Perplexed by this – does baseball have mulligans? – you glance toward the third baseman. He smiles. "Infield fly rule," he says knowingly. You vaguely remember hearing about the infield fly rule, but there's no time to think about it, for the next batter is up and you realize with horror that it's Barry Bonds.

With the first pitch, Bonds swings and connects. It's a hard ground ball – and it's headed straight at you. The ball is moving impossibly fast and it's instinct alone that prods you to put your glove out in its general direction. Smack! The ball takes a bounce and lands in your glove.

No one is more surprised than you are. But there's no time to reflect on your good fortune; the batter on first base is running with all his might toward second base. The second baseman calls out "Two!" as he's headed for second base. Confused, you pick up the ball and weakly throw it to him. He catches it effortlessly, tags the bag, and then fires a strike to first base. It's all happened so fast that you're not sure what occurred, but your teammates are headed to the dugout and one of them pats you on the back telling you that you made a good play. You've survived your first half inning of play. Welcome to the majors.

"You don't set out to use a design pattern; rather you find yourself in a situation where you must make a decision"

If someone were to come up to you at that point and say that you had executed the Double Play design pattern perfectly, you might think them a bit odd. And yet, that's exactly what you did. But you've never studied baseball design patterns – how could you have known what to do? The answer to this question is key to understanding how to use design patterns.

You don't set out to use a design pattern; rather you find yourself in a situation where you must make a decision. What the design pattern does is tell you, "In this situation, the following actions have a higher degree of success than any others we have discovered." In the situation you found yourself in playing shortstop, the Double Play design pattern represented the most successful action you could take.

The Reality Behind the Name

To be a good baseball player, you must study design patterns – learning when to bunt, when to hit a sacrifice fly, or when to steal. The same is true to be a good programmer: you must study design patterns so that you know about such design patterns as Observer, Adapter, and Chain of Responsibility. But just as in baseball, the important thing in software engineering is not so much the name of the design pattern, but the reality behind that name.

René Magritte, commenting on the human tendency to idolize (literally) reality, painted a now-famous picture of a pipe, under which he wrote the words, Ceci n'est pas une pipe: this is not a pipe. The name of a thing is not the thing itself, the map is not the territory, and learning the names of design patterns alone cannot provide guidance on building robust software.

In fact, the names given to design patterns can sometimes make them seem far more mysterious and complex than they are. But design patterns exist only to help you, not to act as a judge on your actions. Mastery involves learning both what design patterns exist and when each is appropriate. Only then can we use what we have learned for our ultimate goal – to craft systems that provide the power and flexibility that both we and our clients want.

If you'd like to learn more about design patterns and in particular their application with ColdFusion Components (CFCs), see the series of articles by Brendan O'Hara, "Design Patterns in ColdFusion," currently running in *CFDJ*, which started in the March 2003 issue. www.sys-con.com/coldfusion/article.cfm?id=577.

About the Author

Hal Helms (www.halhelms.com) is a Team Macromedia member who provides both on-site and remote training in ColdFusion, Java, and Fusebox.

hal@fusebox.org

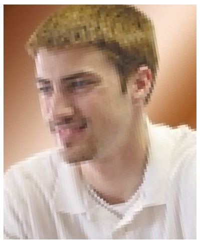
NETQUEST

www.nqcontent.com

Database Searching with Stored Procedures

How to determine which search feature to use

It's easy to program a database search feature, and it's easy to succumb to the temptation to just slam one up as quickly as possible, especially when your manager is standing behind you whispering, "I need this yesterday!" But as developers, we have the responsibility of implementing a search feature as efficiently as possible without making it more complex than necessary.


By Michael Norusis

User satisfaction and infrastructure considerations may be more relevant than upfront coding simplicity. Underneath it all, the real decision here is between a Verity search, a ColdFusion query, or a stored procedure to create a dynamic SQL query. Often, the stored procedure is the best choice.

Recently, I had a project that included creating a public search interface to a membership directory contained in a Microsoft SQL Server 7 database (see Figure 1). The search feature contained some simple criteria from the user's point of view, but to a developer, it could end up being a nightmare.

The requirements called for the ability to conduct a search based on a member's first name, last name, member type, and date range in which the membership was initiated. Additionally, none of the fields were to be required and the user was to be allowed to sort the search results based on a member's name (last, first) or on the date of membership initiation. Clearly, setting up the form was rather simple. It contained a text box for both the first name and last name criteria and a single select drop-down for the member type. Additionally, this simple example uses text fields for searching on the membership date range, but a better solution would be to implement a calendar control (see Listing 1).

What the search form was lacking in complexity, the action page would certainly make up for. Determining how to implement the processing of the directory search would be a little less straightforward. Implementing a Verity solution was ruled out because the user would be unable to easily limit the search results by a date range. There are ways to perhaps support this with custom fields, but even then we need granular searching of each field.

This left using a basic ColdFusion query or writing a stored procedure as the two top contenders. Implementing either option would require using If statements to determine whether or not to include certain SQL AND statements in the WHERE clause of the search query.

Query Processing

When using a typical CFQUERY to search against a database, the initial setup is usually quite straightforward:

```
<CFQUERY DATASOURCE="MyDSN"
                       NAME="SearchQuery">
   SELECT   MemberID, FName, LName,
                      MemberType,
MembershipDate
   FROM      Members
</CFQUERY>
```

Once the initial query setup is complete, the focus then shifts to the WHERE clause of the SQL statement. After all, this is a search mechanism where the user will (almost always) present us with fields to use to filter the results.

Since none of the form fields are required, we cannot assume that each form variable will be needed in the WHERE clause. Therefore, ColdFusion If statements must be included in the query to determine which database fields should be used to limit the search results. See Listing 2, where you'll notice that we've anticipated that the user may not present us with any search fields at all. In such a case we would have no search criteria to use in the WHERE clause. But a frequently used "trick" (when you know you may or may not be adding criteria to the WHERE clause) is to use criteria such as 1=1. Then, you can add more criteria with AND clauses. For example:

```
<CFIF Len(trim(form.Fname))>
   AND
      Members.Fname like
        '%#form.Fname#%'
</CFIF>
```

You may notice I'm not testing for IsDefined("form.Fname"). Some may argue that this is because a text field will always be presented by a form. But what if a user executed the page without coming through the form? You'll see that I've used a series of CFPARAM tags to give all the input fields a default null value, just to avoid this concern.

You'll also notice that I'm using a LIKE clause in this search. If I knew that the end user would enter only exactly matching strings, I'd use an "=" comparison. That would clearly be more efficient. But my client wants us to let the user enter even an initial or portion of a name to be matched. Indeed, we could even teach the users that they could add LIKE pattern characters within the name field they offer. For instance, if they entered P%t%n, that could find several variations of Peterson.

Still other issues you may notice from Listing 2 are the lack of validation of the user's input date values. Specifically, it would be a good idea to test that the "from" date is less than the "to" date, so as not to waste database processing time trying to find a result that will never make sense logically. That would certainly be prudent but is outside the scope of this article and is left as an exercise for the reader.

You may notice that I'm using a string to hold the types in the MemType column. Of course, you could recommend a more normalized design where MemType holds a code from another lookup table. Our focus here isn't on database design, though it's worth noting that if we did that, we could use a query of those values to build the drop-down for member types in the form in Listing 1.

It's also important to note that the ability to search based on the membership date must be handled differently than the other fields. Since none of the form fields are required, the user could enter a date in one of the fields and not the other. As developers, we must anticipate this and account for the possibility in the programming.

If the user enters both dates, the SQL should use a fairly straightforward "BETWEEN" based on those dates. If only one date is entered by the user, we must assume the desired effect is to limit the membership date range based on a minimum or maximum date depending on whether the user enters the "from" date or the "to" date respectively. Adding help text to the form explaining this would certainly be helpful to the user.

Additionally, since SQL Server (and most other databases) can store both the date and time in a date field, we must take this into account. This is done by simply appending "00:00:00" to the from date and "23:59:59" to the end date when it is used in the query.

With the completion of the WHERE clause, our search feature is lacking only the ordering of the query results and a way to display the results (see Listing 2).

Contemplating Stored Procedures

Implementing the search query directly in ColdFusion should not take a novice developer all that long, but it certainly is not the only way to search a database. Using a stored procedure, when available, is a bit more complex and can certainly tack on some extra development time, but should definitely be considered.

The first step is to create the stored procedure, which, for the purpose of this article, will be named uspDirectorySearch.

The second step is to implement the stored procedure through our ColdFusion search template.

Since the search form allows a user to search the membership directory based on five different variables and determine how the results are sorted, we will need to pass six variables into the stored procedure (see Listing 3).

The first step in the procedure is the fixed portion of the query. The SELECT and FROM portions of the SQL statement should be created, because these will not depend on the values entered by the user.

```
SET @BaseSQL = 'SELECT MemberID,
  FName, LName, MemberType,
  MembershipDate FROM Members
  WHERE 1 = 1 '
```

Now we can finally get into the meat of the stored procedure.

Each search criteria variable should be evaluated to determine if an AND statement should be appended to the WHERE clause. Since each variable passed into the stored procedure is NULL by default, check to see if each variable contains search criteria.

```
IF len(@FName) > 0
  SET @SQLStatement = '
    Members.FName like ' + char(39)
    + '%' + @Fname +  '%' + char(39)
```

Once all of our search variables have been checked, we can then use the final variable passed in, to determine how the results should be sorted. Then simply combine the @BaseSQL and @SQLStatement variables and execute the SQL.

```
EXEC (@SQLStatement)
```

Finally, we see how to call the stored procedure from ColdFusion in Listing 4, with a CFSTOREDPROC and its related tags. As with Listing 1, it results in a query resultset called SearchQuery.

It's a rather trivial exercise to create a routine to loop over the resultset and output the records found. Just note that the routine will not differ if the result is created from a stored procedure or a CFQUERY. To ColdFusion, a resultset is a resultset.

Some Side Benefits of Stored Procedures

Obviously, searching the database with a ColdFusion query or with a stored procedure achieves the same end result, but certainly there are times when one option should be used instead of the other. A ColdFusion query is often implemented more quickly and can take less time to modify when changes to the application are necessary. Therefore, if you are assigned a project with a deadline of yesterday and the main requirement is for it to function properly, you may want to use a basic ColdFusion query. However, this ease of implementation comes with a price.

First, when you consider that the database server precompiles each stored procedure in order to execute more quickly, it often becomes clear that using a stored procedure is the best way to go. With SQL provided in a CFQUERY, the database is forced to parse and interpret the SQL being passed each time the query is executed.

There can be security benefits as well. Queries that use data submitted through a form or passed through the URL string can be potential security risks unless they are implemented correctly. This possibility alone is enough to cause some developers to implement each and every database query in a stored procedure. When using a stored procedure, SQL Server (or whichever database you are using) is expecting

MemberID	FName	LName	MemberType	MembershipDate
1	John	Smith	Associate Member	12/18/2000
2	Patrick	McGeehan	Regular Member	12/23/1995
3	Annie	James	Regular Member	11/10/1999
4	Jane	Williams	Associate Member	10/17/2002
5	Eileen	Green	Regular Member	11/2/1997

Figure 1: Membership directory table

certain variables to contain certain types of data and not to exceed certain lengths.

For example, if SQL Server is expecting a variable named "MemberID" to be passed into a stored procedure as a number, and a character is passed in instead, an error is thrown. This can cut down on the number of security vulnerabilities in your code.

Of course, it's worth noting that you can also achieve some measure of the same protection against malformed input with SQL built dynamically in CFQUERY using the CFQUERYPARAM tag. If you're not familiar with that, see the ColdFusion Language reference.

Another benefit of using stored procedures is that the SQL statements are stored in one place, the stored procedure, rather than being buried within a CFQUERY tag in a ColdFusion template.

Implementing stored procedures for your database search interfaces does not just benefit the developer. Systems administrators, database administrators, managers, and help desk personnel often prefer to have developers implement stored procedures (whether they realize it or not).

Applications that are more efficient and help to ensure a more stable server environment are surely administrators' dreams.

By designing and developing your applications so that execution time is minimized and application security is maximized, you are certain to have fewer end user complaints, which should definitely make the managers and help desk personnel at your company very happy.

There are certainly times when implementing a stored procedure is best and other times when a ColdFusion query is warranted. While a full discussion of when to use each of these options is not the purpose of this article, it is important to realize that in almost any development situation you do have a decision as to which to use. Knowing how search features can be implemented in stored procedures is certainly the first step to making the right decision.

About the Author

Michael Norusis is a Macromedia Certified ColdFusion Developer and has been developing applications for four years, integrating ColdFusion with both Oracle and Microsoft SQL Server databases. He has worked as head of special Web projects at The Catholic University of America and is currently a developer at CarFax.com in Fairfax, VA.

michaelnorusis@carfax.com

Listing 1: Public Search Form

```
<FORM NAME="searchForm"
               METHOD="post"
               ACTION="searchAction.cfm">

  Last Name:
  <INPUT TYPE="text"
                 NAME="LName"><br /><br />

  First Name:
  <INPUT TYPE="text"
                 NAME="FName"><br /><br />

  Membership Type:
  <SELECT NAME="MemberType">
    <OPTION VALUE=""></OPTION>
    <OPTION VALUE="Associate Member">
      Associate Member
    </OPTION>
    <OPTION VALUE="Regular Member">
      Regular Member
    </OPTION>
  </SELECT><br /><br />

  Date of Membership Between:

  <INPUT TYPE="TEXT"
                 NAME="DateFrom"
                 MAXLENGTH="10">
  And

<INPUT TYPE="TEXT"
               NAME="DateTo" MAXLENGTH="10">
  <br /><br />

  Sort Results By:
  <INPUT TYPE="RADIO"
               NAME="Sort"
               VALUE="LName, FName">
  Member Name
  <INPUT TYPE="RADIO"
               NAME="Sort"
               VALUE="MembershipDate">
    Date of Membership
  <br /><br />

  <INPUT TYPE="SUBMIT"
               VALUE="Search Now">

</FORM>
```

Listing 2: ColdFusion Search Query

```
<CFPARAM NAME="form.FName" DEFAULT="">
<CFPARAM NAME="form.LName" DEFAULT="">
<CFPARAM NAME="form.MemType" DEFAULT="">
<CFPARAM NAME="form.DateFrom" DEFAULT="">
<CFPARAM NAME="form.DateTo" DEFAULT="">
<CFPARAM NAME="form.Sort" DEFAULT="Name">

<CFSET variables.From = trim(form.DateFrom)>
<CFSET variables.To = trim(form.DateTo)>

<CFQUERY DATASOURCE="GroupLogic_Content"
                       NAME="SearchQuery">
  SELECT  MemberID, FName, LName,
                          MemberType, MembershipDate
  FROM    Members
  WHERE   1 = 1
<!--- If searching on last name --->
  <CFIF Len(trim(form.Lname))>
    AND Members.Lname like '%#form.Lname#%'
  </CFIF>

<!--- If searching on first name --->
```

IVIS TECHNOLOGIES

www.ivis.com

```
  <CFIF Len(trim(form.Fname))>
    AND Members.Fname like '%#form.Fname#%'
  </CFIF>
<!--- If searching on member type --->
  <CFIF Len(trim(form.MemberType))>
    AND Members.MemberType = '#form.MemberType#'
  </CFIF>
<!--- If searching on both from and to date --->
  <CFIF Len(variables.From) AND Len(variables.To)>
    AND (Members.MembershipDate BETWEEN
    '#variables.From# 00:00:00' AND
    '#variables.To# 23:59:59')
<!--- If searching on only from date --->
  <CFELSEIF Len(variables.From) AND
   NOT Len(variables.To)>
    AND (Members.MembershipDate >=
    '#variables.From# 00:00:00')
<!--- If searching on only to date --->
  <CFELSEIF NOT Len(variables.From)
   AND Len(variables.To)>
    AND (Members.MembershipDate <=
    '#variables.To# 23:59:59')
  </CFIF>

  ORDER BY #form.Sort# DESC
</CFQUERY>
```

Listing 3: SQL Server Stored Procedure

```
CREATE PROCEDURE uspDirectorySearch (
   @FName   varchar(75)  =  NULL,
   @LName   varchar(75) = NULL,
   @MemberType  varchar(75) = NULL,
   @DateFrom varchar(11) = NULL,
   @DateTo  varchar(11) = NULL,
   @Sort  varchar(15) = 'LName, FName')

AS

DECLARE
   @BaseSQL varchar(1000),
   @SQLStatement  varchar(8000)

/* Initialize the base SQL*/
SET @BaseSQL = 'SELECT MemberID, FName,
   LName, MemberType, MembershipDate
   FROM Members WHERE 1 = 1'
SET @SQLStatement = ''

/* If searching on first name */
IF len(@FName) > 0
   SET @SQLStatement = 'WHERE Members.FName
   like ' + char(39) + '%' + @FName + '%' + char(39)

/* If searching on last name */
IF len(@LName) > 0
   SET @SQLStatement = @SQLStatement + ' AND
   Members.LName like ' + char(39) + '%' + @LName
   + '%' + char(39)

/* If searching on member type */
IF len(@MemberType) > 0
   SET @SQLStatement = @SQLStatement + ' AND
   Members.Membertype = ' + char(39) + @MemberType
   + char(39)

/* If searching with both dates */
IF (len(@DateFrom) > 0  AND len(@DateTo) > 0)
   SET @SQLStatement = @SQLStatement + 'AND
   (Members.MembershipDate BETWEEN ' + char(39)
   + @DateFrom + ' 00:00:00' + char(39) + 'AND '
   + char(39) + @DateTo + ' 23:59:59' + char(39) + ')'
/* If searching with only the From date*/
ELSE IF (len(@DateFrom) > 0 AND len(@DateTo) < 1)
   SET @SQLStatement = @SQLStatement
   + ' AND (Members.MembershipDate >= ' + char(39)
   + @DateFrom + ' 00:00:00'  + char(39) + ')'
/* If searching with only the To date */
ELSE IF (len(@DateFrom) < 1 AND len(@DateTo) > 0)
   SET @SQLStatement = @SQLStatement
   + ' AND (Members.MembershipDate <= ' + char(39)
   + @DateTo + ' 23:59:59' + char(39) + ')'

/* Append the order by clause */
SET @SQLStatement = @BaseSQL +  @SQLStatement
   + ' ORDER BY ' + @Sort + ' DESC'

/* Execute the SQL */
EXEC (@SQLStatement)
```

Listing 4: Calling the Stored Procedure

```
<CFPARAM NAME="form.FName" DEFAULT="">
<CFPARAM NAME="form.LName" DEFAULT="">
<CFPARAM NAME="form.MemberType" DEFAULT="">
<CFPARAM NAME="form.DateFrom" DEFAULT="">
<CFPARAM NAME="form.DateTo" DEFAULT="">
<CFPARAM NAME="form.Sort" DEFAULT="LName, FName">

<CFSTOREDPROC DATASOURCE="GroupLogic_Content"
PROCEDURE="uspDirectorySearch">
  <CFPROCPARAM TYPE="IN"
    VALUE="#FORM.FName#"
    DBVARNAME="@FName"
    CFSQLTYPE="CF_SQL_VARCHAR">

  <CFPROCPARAM TYPE="IN"
    DBVARNAME="@LName"
    VALUE="#FORM.LName#"
    CFSQLTYPE="CF_SQL_VARCHAR">

  <CFPROCPARAM TYPE="IN"
    VALUE="#FORM.MemberType#"
    DBVARNAME="@MemberType"
    CFSQLTYPE="CF_SQL_VARCHAR">

  <CFPROCPARAM TYPE="IN"
    VALUE="#FORM.DateFrom#"
    DBVARNAME="@DateFrom"
    CFSQLTYPE="CF_SQL_VARCHAR">

  <CFPROCPARAM TYPE="IN"
    VALUE="#FORM.DateTo#"
    DBVARNAME="@DateTo"
    CFSQLTYPE="CF_SQL_VARCHAR">

  <CFPROCPARAM TYPE="IN"
    VALUE="#FORM.Sort#"
    DBVARNAME="@Sort"
    CFSQLTYPE="CF_SQL_VARCHAR">

  <CFPROCRESULT NAME="SearchQuery" />
</CFSTOREDPROC>
```

Download the Code...
Go to www.coldfusionjournal.com

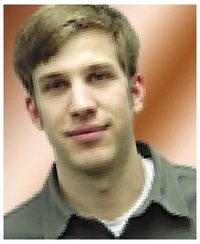
TERATECH

www.cfconf.org/cfun-03/

Back to Basics

Simplify text file parsing in ColdFusion

I read somewhere that when faced with a task that takes one hour to do manually, or one hour to automate, a good programmer will choose to automate the process. As ColdFusion developers, we often face this decision when we need to programmatically use data contained in a text file.



By Christian Thompson

There are two ways to access this data – automating the process by parsing the text file or manually inputting the data. This article presents some basic techniques that can help you make the choice to be a "good programmer" by automating the process.

Bulk upload and data import are the two primary types of functionality in which text file parsing is used in Web application development. The main difference between these two is the format of the text file being parsed. Bulk upload functionality parses text files designed specifically to be processed programmatically, while data import functionality tries to parse files created for human consumption. To handle these differences, slightly different techniques are required.

Bulk Upload

Figure 1 shows an example of a file created for bulk upload. It contains a listing of new employees who need to be added to an employee database. Because this file was created specifically for computer processing, it was created using a table structure. The first row contains the column names, and each subsequent row contains one employee record with name, phone, and e-mail address columns.

```
Name,Phone,Email
John Doe,555-555-5555,john@acompany.com
Jane Doe,444-444-4444,jane@acompany.com
```

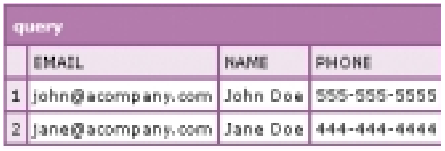
Figure 1: c:\BulkUpload.txt

The table structure makes parsing this file pretty easy with either the cfloop tag or the cfhttp tag. To use the cfloop tag, first read the file into a variable (str_Content in the example below) using the cffile tag and then loop over the content using the correct delimiter (carriage return/line feed – chr(13)chr(10) – on Microsoft Windows and line feed – chr(10) – on Unix). Use the listGetAt() function to access specific fields in each line.

```
<cfset bln_FirstLine = true>
<cfloop
  list="#str_Content#"
  delimiters="#chr(13)##chr(10)#"
  index="str_Line">

  <!--- Ignore the column name
    line --->
  <cfif bln_FirstLine is false>
    <cfset str_Name =
      listGetAt(str_Line, 1, ",")>
    <cfset str_Phone =
      listGetAt(str_Line, 2, ",")>
    <cfset str_Email =
      listGetAt(str_Line, 3, ",")>
  <cfelse>
    <cfset bln_FirstLine = false>
  </cfif>
</cfloop>
```

For rigidly formatted documents accessible via HTTP, the cfhttp tag makes parsing the document even easier. Given the right parameters, this tag will automatically parse the content and return a query object containing the results. The following example produces the output shown in Figure 2:

```
<cfhttp method="GET"
  url="#dir_URL#/bulkupdate.txt"
  name="qry_Contents"
  delimiter=","
  textQualifier="">
</cfhttp>

<cfdump var="#qry_Contents#">
```



query

	EMAIL	NAME	PHONE
1	john@acompany.com	John Doe	555-555-5555
2	jane@acompany.com	Jane Doe	444-444-4444

Figure 2: CFHTTP results

Data Import

```
John Doe
Phone: 555-555-5555

Jane Doe
Phone: 444-444-4444
```

Figure 3: c:\DataImport.txt

Figure 3 shows an example of a file that was not created to be used programmatically. It is an employee directory designed to be viewed by humans. Parsing files like this one is more complicated because each line cannot be processed in the same way. The logic must first determine what information is contained in the current line and then process the line accordingly. One way to do this is to track the current line type with a variable. After each line is processed, you should be able to infer the type of line that will be processed next, based on the current line type, and set the variable accordingly.

For example, when processing the line containing the employee's name, you know the next line will contain the employee's phone number. Therefore, after processing the name line, you set the line type equal to "phone" and loop. On the next loop, the appropriate logic processes the phone line, sets the line type back to "name," and the process repeats.

```
<!--- The first line contains
  the employee's name --->
<cfset str_LineType = "name">

<cfloop list="#str_Content#"
  delimiters="#chr(13)##chr(10)#"
  index="str_Line">

  <cfif str_LineType is "name">
    <cfset str_Name = str_Line>
    <cfset str_LineType = "phone">
  <cfelseif str_LineType is "phone">
    <cfset str_Phone =
      listGetAt(str_Line, 2, ": ")>
    <cfset str_LineType = "name">
  </cfif>
</cfloop>
```

CFML Parsing Functionality Limitations

As the examples above show, ColdFusion has several powerful built-in functions and tags that you can use for parsing, such as listGetAt(), cfhttp, and cfloop. Unfortunately, these functions share a common limitation: they treat consecutive delimiters as one delimiter. For example, the built-in ColdFusion functions consider the line "Jim Doe,,jim@acompany.com" to have only two tokens ("Jim Doe" and "jim@acompany.com"), even though the strings are separated by two commas.

This was not a problem in the above examples because there were no empty fields in the data being processed. If the data did have empty fields, however, the example code would process it incorrectly. Consider the result of parsing the line "Jim Doe,,jim@acompany.com" with the bulk update code in the first example. Since the listGetAt function treats consecutive delimiters as one delimiter, the code would set str_Phone equal to "jim@acompany.com" and str_Email equal to a blank string. Obviously, this is incorrect.

My Solution

I have developed a ColdFusion tag to address this shortcoming: TextParse.cfm (Listing 1). The TextParse tag treats consecutive delimiters as delimiters surrounding an empty string. Therefore, it considers "Jim Doe,,jim@acompany.com" as having three tokens – "Jim Doe", "", and "jim@acompany.com" – rather than two.

The code implementing the TextParse tag is relatively straightforward. From a high level, it simply loops over the delimiters in the content, and extracts the strings that fall between the delimiters. This logic is actually done twice, once in the tag's body and once in the function TokenizeLine(). The tag body breaks the content into separate lines and TokenizeLine() then breaks each line into tokens.

Like cfloop, the TextParse tag is used by placing the code to process each line between its start and end tags. The tag takes three parameters: str_Filename, str_LineDelimiter, and str_TokenDelimiter. The variable "str_Filename" expects the full path of the file to be parsed. "str_LineDelimiter" allows you to specify the delimiter used to separate the lines in the file (by default "#chr(13)##chr(10)#"), and "str_TokenDelimiter" allows you to specify the delimiter to separate the tokens in each line (by default ","). The TextParse tag returns two variables to the caller scope: TextParse.str_Line and TextParse.ar_Tokens. TextParse.strLine is a string containing the complete current line and TextParse.ar_Tokens is an array containing the tokens of the current line.

The following example demonstrates the use of the TextParse tag to parse BulkUpload.txt. Because text files often have inconsistent formats, I wrapped the token array accesses in a cftry/cfcatch statement. Without the error handling, the example code would generate an error when processing a line with fewer than expected fields. Although this seems to complicate things, it's good to know when a line doesn't have the right format, since your parsing logic might otherwise handle it incorrectly. The error handling allows you to flag the line for later examination, and continue processing.

```
<cf_TextParse
  str_Filename="c:\BulkUpload.txt">

  <cftry>
    <cfif TextParse.ar_Tokens[1]
      is not "Name">
```

SOUTHWEST SUPPORT SOLUTIONS

```
      <cfset str_Name =
        TextParse.ar_Tokens[1]>
      <cfset str_Phone =
        TextParse.ar_Tokens[2]>
      <cfset str_Email =
        TextParse.ar_Tokens[3]>
    </cfif>

    <cfcatch>
      <!--- Handle error --->
    </cfcatch>
  </cftry>
</cf_TextParse>
```

Conclusion

ColdFusion developers can use text file parsing to import data meant for human consumption and to allow Web site users to make bulk uploads. Using text file parsing effectively requires knowledge of basic techniques as well as the limitations of ColdFusion's parsing functionality. Armed with the techniques presented above and the TextParse tag, the decision to automate data import and bulk upload processes should be an easier one. Having made the choice to automate, you can then be confident in your status as a "good programmer."

About the Author

Christian Thompson is a certified advanced Macromedia ColdFusion MX developer. He is a senior software engineer for Inserso, a technology consulting firm headquartered in Annandale, VA, where he has specialized in ColdFusion application development for over two years.

cthompson@inserso.com

Listing 1

```
<cfparam name="attributes.str_Filename" default="">
<cfparam name="attributes.str_LineDelimiter"
  default="#chr(13)##chr(10)#">
<cfparam name="attributes.str_TokenDelimiter"
  default=",">

<cffunction name="TokenizeLine" returntype="array">
  <cfargument name="str_Content" type="string"
     required="true">
  <cfargument name="str_Delimiter" type="string"
    required="true">

  <cfset var ar_Tokens = arrayNew(1)>
  <cfset var int_CurrPos = 1>
  <cfset var int_NextPos = 1>
  <cfset var str_Token = "">

  <cfscript>
    int_NextPos =
      REFind("#arguments.str_Delimiter#|$",
      arguments.str_Content, int_CurrPos);
    while (int_NextPos gt 0)
    {
      str_Token = mid(arguments.str_Content,
        int_CurrPos, int_NextPos - int_CurrPos);
      arrayAppend(ar_Tokens, str_Token);
      int_CurrPos = int_NextPos +
        len(arguments.str_Delimiter);
      int_NextPos =
        REFind("#arguments.str_Delimiter#|$",
        arguments.str_Content, int_CurrPos);
    }
    // If the line ends with a token, add
    // an extra empty element to the array
    if (len(arguments.str_Content) gt 0
      and mid(arguments.str_Content,
        len(arguments.str_Content), 1)
        is arguments.str_Delimiter)
    {
      arrayAppend(ar_Tokens, "");
    }
  </cfscript>
  <cfreturn ar_Tokens>
</cffunction>

<cfif ThisTag.ExecutionMode IS "Start">
  <!--- Read the file and inititialize position
    and caller variables --->
  <cffile action="read"
    file="#attributes.str_Filename#"
    variable="str_Content">
  <cfset int_CurrPos = 1>
  <cfset caller.TextParse.str_Line = "">
  <cfset caller.TextParse.ar_Tokens = arrayNew(1)>
</cfif>

<!--- Find the next delimiter. The regular
  expression "#attributes.str_LineDelimiter#|$"
  finds the next line delimiter OR the end of
  the string. --->
<cfset int_NextPos =
  REFind("#attributes.str_LineDelimiter#|$",
  str_Content, int_CurrPos)>

<cfif int_NextPos gt 0>
  <!--- Get and return the line and tokens --->
  <cfset str_Line = mid(str_Content,
    int_CurrPos, int_NextPos - int_CurrPos)>
  <cfset caller.TextParse.str_Line = str_Line>
  <cfset caller.TextParse.ar_Tokens =
    TokenizeLine(str_Line,
    attributes.str_TokenDelimiter)>

  <!--- reset the current position --->
  <cfset int_CurrPos = int_NextPos +
    len(attributes.str_LineDelimiter)>
</cfif>

<cfif ThisTag.ExecutionMode IS "End">
  <!--- Keep looping as long as we have
    content to process --->
  <cfif int_NextPos gt 0>
    <cfexit method="Loop">
  </cfif>
</cfif>
```

Download the Code...
Go to www.coldfusionjournal.com

MACROMEDIA

www.macromedia.com/go/certification1/

Real Time = Real Problem

It's up to developers to make the choice

Most Web-based applications operate in real time. Add an article to a database and it shows up immediately on content pages. Update a user address and the new contact information is available immediately. Add or remove an employee and the phone directory is correct when next viewed. Real-time data in a real-time world. That's a good thing, isn't it?

By Ben Forta

Real Time Is Real Expensive

In an ideal world, real-time everything would indeed be a good thing. But we don't live in an ideal world. As appealing as always being up-to-date is, real time comes with a real cost:

- ***Performance:*** To put it quite simply, nothing eats up performance as much as real-time processing. The more dynamic your application, the worse it will perform. After all, static pages don't suffer from performance problems, ever.
- ***Scalability:*** An extension of the above, the more real-time processing your application performs, the less it will scale. If you make your applications work harder they'll just be able to do less concurrently, there's no way around that one.

If you were to analyze all the timings and debug output from your ColdFusion applications, you'd undoubtedly find that more time is spent processing <CFQUERY> tags than anything else (possibly even everything else combined). In other words, eliminate all your <CFQUERY> tags and your application will fly. And while avoiding databases is not at all practical (or even advisable), understanding the price of real-time data interaction is important.

Of course, database access and <CFQUERY> are not the only culprits; you likely use Web services and CFX tags and calls to COM and Java and more, and all of those can impact performance too. But databases are a good place to start because they're so prevalent and because making changes (where appropriate) is actually not that difficult.

Is Real Time Really Necessary?

Obviously, some applications (or parts thereof) must be real time. Could you imagine eBay auctions if bids were updated only periodically? How do you think users would react to Amazon.com listing products as in stock only to send out an "oops" e-mail after checkout? What would users do if they changed their AOL or Yahoo or MSN passwords only to find that the password change took some unknown length of time to take effect? All of these sites utilize real-time processing to ensure the best customer experience – some operations simply must occur in real time.

And that's key – some operations, but not all. If you were to place an online classified listing on any of the major classified sites (including Yahoo) you'd find that your ad did not appear instantaneously. Rather, listings are updated with current information at regular intervals. Similarly, online user directories obtain new data on a regular basis, but online listings sometimes take months to be updated.

So why are *some* operations performed in real time and others not? Simply because there is a tradeoff to be made – the more data is real time, the greater the hit on performance and scalability. As such, developers of applications have to choose between the two, and for many, the choice is to not perform real-time processing unless it is absolutely required.

But most ColdFusion developers don't make the choice at all. ColdFusion makes implementing real-time processing so easy (much easier than implementing anything non–real time) that they go the real-time route by default. That's a real problem.

Reducing Database Reads

I'm not going to be able to cover every real-time scenario in this column, but I would like to point out some ideas you should think about as a starting point. The simplest (and remarkably effective) change you can make to your application involves the reading of data from database tables. <CFQUERY> is a very powerful tag; it lets you access all sorts of databases easily, maybe too easily. And so developers tend to overuse

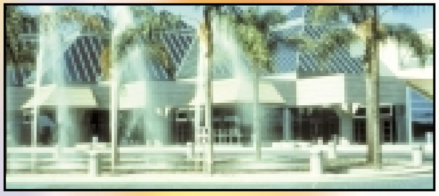
International Web Services Conference & Expo

Web Services Edge WEST 2003

web services EDGE conference&expo

SEPT. 30 - OCT. 2, 2003

Santa Clara, CA



EXTENDING THE ENTERPRISE WITH WEB SERVICES THROUGH JAVA, .NET, WEBSPHERE, MAC OS X AND XML TECHNOLOGIES

Featured technologies and topics will include:

- Focus on Java
- Focus on .NET
- Focus on WebSphere
- Focus on Mac OS X
- Focus on XML

XML

WebSphere

Microsoft .net

Mac OS X

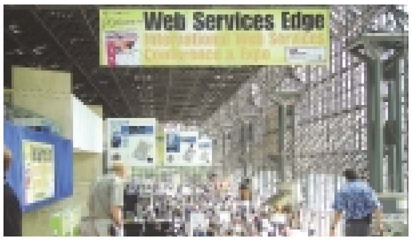
LINUX EDGE conference&expo

web services EDGE conference&expo

BOSTON

February 24-27, 2004

For more information visit
www.sys-con.com
or call
201 802-3069



Over 100 participating companies will display and demonstrate over 300 developer products and solutions.

Over 2,000 Systems Integrators, System Architects, Developers, and Project Managers will attend the conference expo.

Over 60 of the latest sessions on training, certifications, seminars, case studies, and panel discussions will deliver REAL World benefits, the industry pulse and proven strategies.

Contact information: U.S. Events: 201 802-3069 or e-mail grisha@sys-con.com

WEB SERVICES EDGE WEST 2003
CALL FOR PAPERS NOW OPEN
Submit your papers online at:
www.sys-con.com/webservices2003west

PRODUCED BY SYS-CON EVENTS

WebServices Journal · JAVA Developer's Journal · WebSphere Developer's Journal · XML Journal · .NET Developer's Journal · SAMS · BEA WebLogic Developer's Journal · wireless Business & Technology · LINUX Business & Technology · CF Advisor · ColdFusion Developer's Journal · PowerBuilder Developer's Journal

Java is a registered trademark of Sun Microsystems, .NET is a registered trademark of Microsoft, Mac OS X is a registered trademark of Apple Computer, Inc., WebSphere is a registered trademark of IBM. All other product names herein are the properties of their respective companies.

<CFQUERY>, often rereading data that likely has not changed (or has changed with changes that need not be utilized immediately).

I covered reducing database access via caching extensively in a column entitled “Caching in on Performance” (***CFDJ,*** Vol. 1, issue 2). As explained there:

Where would you use caching within your applications? Here are some examples:

- Almost every form that prompts for an address displays a list of states. Those states should never be hard coded (even though there’s no 51st state scheduled to join the U.S. at this time); instead, state lists should be populated by a query against a states table. But as that states list doesn’t change often (it’s 40 years since Hawaii came on board), reading it from the database every time it’s needed is a waste of database resources. The states list is thus a primary candidate for caching.
- Employee lists are another good example. While it’s true that employee lists can change frequently, it’s doubtful that they change so often that they have to be read from the database each time they’re needed (if they do, do yourself a favor: find a new employer, and quickly). Caching employee lists for a few hours will reduce database activity, and the only penalty is that personnel changes won’t be immediately reflected in your lists.

Even though frequently retrieved data is likely cached by the database server itself, retrieving the data again is obviously more resource intensive than not requesting it at all. Furthermore, as ColdFusion usually isn’t running on the same box as the database server, eliminating unnecessary database requests can also reduce network traffic between the two machines, which in turn further eliminates potential performance bottlenecks.

ColdFusion provides two different ways to cache database reads:

- Query-based caching using CACHEDWITHIN.
- Variable-based caching in which queries are stored in persistent scopes.

“Real time has become the norm by default, not by necessity. And real time causes real problems”

I am not going to explain these here; refer to the previously mentioned column to learn more.

Reducing Dynamic Processing

Beyond database access, you likely have entire blocks of your application that are being generated programmatically in real time, but that perhaps need not be. For example, the above mentioned employee list. Not hitting the database unnecessarily is a great first step, but you also loop through the results creating output and embedding formatting. Does that really need to occur on each and every page request?

ColdFusion provides several options that may be used to reduce dynamic processing, and you may use any or all of them (or roll your own). In order of granularity:

- <CFSAVECONTENT> can be used to save the results of any processing to a variable, perhaps a variable in a persistent scope. This allows developers to mark blocks of code that are executed after timeouts or on specified intervals. Using <CFSAVECONTENT> it is possible to cache Web services results, file reads, returned parsed XML, and much more.
- <CFCACHE> can be used to cache entire pages so that the generated page output is saved and served up on future requests until a specified timeout. Using <CFCACHE> an entire page can be returned in what is effectively a <CFFILE ACTION=“read”> and a few other CFML statements.
- It is also possible to save generated CFM pages as static HTML files so that no application server processing need occur at all at runtime.

Of course, each of these options requires that you give something up; if you serve cached content you are serving old (not real-time) content. But depending on what your app is, that may be entirely acceptable. And if so, all you have to lose are performance and scalability problems.

Using Delayed or Batch Processing

One of the most important (and least trivial) concepts in the real-time discussion is the use of delayed or batch processing. The best way to understand the idea is via examples. So:

- You need to import data from a text file into your database. You know that reading the file and then inserting (or updating) each row using a <CFQUERY> within a <CFLOOP> is slow and highly error prone. So you use a batch upload utility (like SQL Server’s bcp) to dump all the data into a new empty table. A scheduled event on the database server checks for data in this table, and if any is present, fires off a stored procedure that reads each row, validates it, breaks it up into the appropriate relational components, and then performs the database INSERT or UPDATE operations as needed.
- You want contact information in your database tables to be clean and consistent – all states are two-letter abbreviations in upper case (with no trailing period), all names are first letter capped, titles like Mr. and Mrs. must have a trailing period, phone numbers must be formatted in a specific way, leading and trailing spaces on all fields must be removed, and so on. Initially you did all that cleanup in CFML before the SQL INSERT, but

then you realized that in doing so you were not only hurting performance, you also were not cleaning up any data that did not originate in a ColdFusion application. And so you change the app so that data is written as is, and whenever an INSERT or UPDATE occurs, you set a row level flag named DIRTY to true. You then create a database scheduled event that runs once a day and performs all the cleanup for any rows where DIRTY is true, and then upon completion sets DIRTY to false to flag rows as clean.

- Your e-commerce site allows users to pay by credit card. You know that most credit card transactions fail because of poor data entry (bad number or expiration date, for example) and so you do basic error checking to ensure that the credit card number is valid post data entry. But you do not actually submit the credit card information for approval while the user waits. Rather, using the assumption that most credit card transactions do not fail (especially from known repeat customers) you thank the customer for the order and place the credit card transaction in a queue. You'll notify the customer via e-mail only if there is a problem. This ensures that the application is responsive. It can also prevent double billing (which could occur if a customer were to submit a form twice), and creates a better user experience. (FYI, you may be surprised to know that some of the largest e-commerce sites on the Net do just this.)

In all of these examples, some processing is postponed and/or batched. The result? Not only are the applications faster and more scalable, but the developers also have greater control over exactly what operations occur and when.

Conclusion

Real time has become the norm by default, not by necessity. And real time causes real problems. Database caching, dynamic output caching, and delayed or batch processing are all concepts that can (and should) be leveraged so as to improve application performance and scalability. The truth is, there is no right or wrong here – everything is a tradeoff. Not all options will always be usable (you'd not want to use delayed batched credit card processing, for example, if you are selling access to a paid Web site). As a developer you get to make the real-time versus non-real-time choice. The important thing is that you actually make the choice. And I think you'll find that most parts of most applications actually need not function in real time at all.

About the Author

Ben Forta is Macromedia's senior product evangelist and the author of numerous books, including ColdFusion MX Web Application Construction Kit *and its sequel,* Advanced ColdFusion MX Application Development, *and is the series editor for the new "Reality ColdFusion" series. For more information visit www.forta.com.*

ben@forta.com

CFDYNAMICS

www.cfdynamics.com

Macromedia's Data File Access API Architecture Unleashed

XML for the real world

Like its predecessors, Macromedia's most recent installment of the Devnet Resource Kit (DRK 3) is stocked with many excellent utilities for Flash developers. Unlike previous releases, DRK 3 aims to make the lives of ColdFusion developers easier by including many applications and development tools for use in CFMX applications. One of these is an Application Programming Interface I developed called the Data File Access API (DFA API).

By Simon Horwith

When I set out to design and develop the DFA API, the goal was to develop an API that would allow developers to store data in text files as either XML or CSV text, and to access and manipulate that data as easily as if it were stored in a database. With the goal of that core functionality in mind, the primary objective was to invent an architecture that would perform as optimally as possible.

In addition, two other primary objectives were to make the API very easy to use and to make it flexible enough for developers to extend or implement in any way they might need. Ideally, as developers become more familiar with the API, they will be inspired to use it as the backbone of more creative solutions to meet their applications' needs. Let's examine how the features of ColdFusion MX were used to meet these objectives.

The first thing I needed to consider was how to architect the API not only to define and store data, but also to make this data available for very fast filtering and retrieval. What I decided was to create a ColdFusion Component that houses all of the methods for working with the data and that stores all of the data in memory (as needed) in a proprietary XML DOM format, whether the data came from CSV or XML text. I refer to these as "data tables" and think of them as being analogous to database tables cached in memory.

The API needs to be flexible enough to allow data to be retrieved and filtered using XPath or SQL, so component methods exist to determine whether the query passed is XPath or SQL. XPath is applied directly to the XML DOM in memory and in order to use SQL, the XML DOM is first converted to a ColdFusion query object and then queried using Query of Queries. Any call to the API to extract data can retrieve that data as XML or as a ColdFusion query.

Another challenge in developing the API was how to physically define and store the data used in applications. I broke the data table definition task between three XML files: one that defines table definitions (column names, data types, default values, etc.); one that maps the definitions to the actual storage locations (as relative or absolute path or URL) so that the API would know where to find the data; and one XML file that contains the data itself. I chose this architecture so that developers can easily write validation routines (the data type and required properties of data table columns aren't actually used), share data table definitions between applications, etc.

In addition to the ability to retrieve data mentioned above, methods were also written to parse SQL statements that perform INSERT, UPDATE, or DELETE operations on data tables. Unlike when a SELECT statement is passed to the API, converting the XML data table to a ColdFusion query object will do no good for INSERT, UPDATE, and DELETE commands, as ColdFusion does not support these SQL statements in a Query of Queries.

Instead, the SQL is broken into its various components and then executed against the appropriate XML nodes directly. In the case of DELETE commands, working with the data directly as XML proved more efficient than converting the XML to a query object, retrieving the data not being deleted, and converting the new query object back to XML.

In addition to working with XML, the API needed to support CSV so a method was added to parse CSV text and convert it to an XML table in memory. The first row of values in the CSV content is used to create a data table definition, and all other rows populate that definition. Other methods were also added for validation of various entities being used, to make debugging easier, etc. Two other major concerns while developing the API were how to create an easy way for all developers to use the API, and how to handle concurrency issues.

In order to deal with data table memory and physical file concurrency issues, all data retrieval is performed within "read-only" named locks. When row(s) of data are inserted, updated, or deleted from a data table, the data table in memory is first manipulated within an "exclusive" named lock. Afterwards, the entire data table is written to file as an XML string, also within an "exclusive" named lock. This approach minimizes locking on the server, and prevents developers from having to lock API access in their applications, because the API is handling all of the locking – local to the code blocks that require locks.

To make the API easier for developers to use, I wrote a custom tag "wrapper" for the API CFC. The idea behind the custom tag was to give users the ability to use syntax similar to what they are already used to with the <cfquery> tag in order to query the DFA API data for their application. Like <cfquery>, when retrieving data a "name" attribute is passed to assign a name to the result set returned (may be in query or XML format). A "returntype" attribute is used to specify whether to return the results of a query as XML or a ColdFusion query object.

Also similar to <cfquery>, the SQL to SELECT data is passed as the contents of the tag. An XPath attribute is used to select data using an XPath query. Rather than passing a "datasource" name, the tag accepts a "datatable" attribute in order to determine what data table to apply XPath queries to (SQL queries simply name the data table in the SQL). There is also an "XSLT" attribute for performing XSL Transformations on the data (the attribute value is either the location or contents of an XSL stylesheet) and a "CSV" attribute for passing the location of a CSV file (or CSV content) to be parsed into a data table.

Within the tag body a SQL INSERT, UPDATE, SELECT, or DELETE command may be passed, as well as a DROP command to remove a data table from memory, and a SAVE command for committing a data table already in memory to file. The tag itself creates a DFA API instance in the application scope if one doesn't already exist (in start mode), and performs all of its "work" in end mode. The only thing required to use the tag is the existence of three request scope variables that store the locations of the DFA API component, the location of the XML file that defines data table "columns," and the location of the XML file that "maps" these definitions to physical files.

Though the API was never intended for use in large enterprise-level applications, early tests have yielded surprising performance results. The API definitely does perform very well…exactly how much data or how many concurrent users is too many is something you'll have to test for yourself. I wouldn't be surprised to find that even large-scale solutions can be delivered, driven by the API rather than by a traditional database. Even if you decide to stick with the more traditional methods of data storage, I highly recommend looking to the DFA API to serve as an example of how to best architect an API and as an example of how ColdFusion Components, Custom Tags, Query of Queries, and XML Parsing functionality in ColdFusion MX can be combined to achieve amazing results in your applications.

About the Author

Simon Horwith has been using ColdFusion since version 1.5. He is a Macromedia Certified Advanced ColdFusion and Flash developer and is a Macromedia Certified instructor. In addition to administering the CFDJList List Serve and presenting at CFUGs and conferences around the world, he has also been a contributing author of several books and technical papers. Originally from the Washington, DC area, Simon is currently working as a private consultant in London, England.

simon@horwith.com

HOSTMYSITE.COM

www.hostmysite.com

Getting into HomeSite+

ColdFusion Studio fans, listen up!

Fans of ColdFusion Studio may lament that Macromedia no longer sells the product, but take heart. It really does continue to exist in the guise of HomeSite+, which is basically the same thing and then some. What's more – if you own Dreamweaver MX, you can install HomeSite+ free. Have you checked it out?

By Charlie Arehart

Whether you're a current Studio user who's happy where you are or wish you could simply add CFMX tag support to Studio 4.5 or 5 (you can), or someone who looked into HomeSite+ and found it disappointing (which may no longer be the case), this article is for you.

And if you or your organization moved to Dreamweaver MX and/or Studio MX, you may have thought that you had to give up CF Studio. Many folks don't even realize they can install HomeSite+ free if they have Dreamweaver MX.

Yes, I know there are some beefs with HomeSite+ compared with Studio, and those have been addressed in an updater (yes, an updater for HomeSite+). Indeed, the updater fixes many bugs that have long challenged CF Studio, so in essence this updater can be considered an update to CF Studio. Who said Macromedia left us CF Studio users high and dry? (The updater works for HomeSite+, though not CF Studio.)

The cost of upgrading from Studio 4.5/5 to Dreamweaver MX (and HomeSite+) is very reasonable. If you're still using CF Studio 4.5, installing HomeSite+ is an easy way to get an instant upgrade to (and over) CF Studio 5.

In this article, I'd like to introduce you to HomeSite+, tell you how to get it and also how to get the updater, and how to update CF Studio 4.5/5 for CFMX tag support.

(From now on, when I refer to "Studio" I'll mean ColdFusion Studio, not Studio MX, which is the packaging of Dreamweaver MX, Flash MX, Fireworks MX, Freehand MX, and CFMX Developer Edition.)

This isn't an introduction to the various features of HomeSite+ (and Studio), just an explanation of getting into it, especially for current users of CF Studio. If you're new to Studio/HomeSite, I'll point you to some resources here, and introduce more about it in a future article (such as RDS, using Development Mappings to do browsing and debugging, and more).

I've been a big fan of Studio for many years (and of HomeSite+ since it came out with Dreamweaver MX). I frequently share all these points and more with folks in my blog (cfmxplus.blogspot.com) and my user group presentations, so it seemed that an article would be a good opportunity to capture all the points about getting into HomeSite+ for Studio users.

By the way, I'm not suggesting that Dreamweaver MX users abandon it in favor of HomeSite+. Indeed, I'll talk about integrating the two. For those Studio users who have been reluctant to get into Dreamweaver MX, I'll note that it really can be made to work in a way that suits Studio users, including solving problems that make it appear to perform very slowly. I've discussed it in a series of blog entries, such as http://cfmxplus.blogspot.com/2002_11_10_cfmxplus_archive.html#85665430 and http://cfmxplus.blogspot.com/2002_09_22_cfmxplus_archive.html#85498840.

What Is HomeSite+?

There's some debate about how best to characterize HomeSite+. One might assert that it's just Studio 5 with some minor updates applied, renamed due to the conflict with Studio MX. It sure looks like Studio, with just a change in the title bar look (see Figure 1), and for Studio fans that's not such a bad thing. At least it still lives on in this new guise.

Notice that it has the same resource tabs (including the database tab) and the debugger toolbar at the bottom, as well as quickbars at the top right for CF features, and color coding of CF tags as shown in the code example displayed. Indeed, for folks skipping from Studio 4.5, notice that Studio 5 added – and HomeSite+ continues offering – not one but two file tabs (labeled 1 and 2) in the lower left of the resource toolbar. This is great for having a tab for your local and remote files, for instance, or for two projects.

Why then would some consider it to be a step down from Studio 5? Well, about the time Macromedia was preparing to release it, Adobe won a lawsuit that forced some changes to the interface. (See the 5/22/02 Fusion Authority article at www.fusionauthority.com/alert/index.cfm?alertid=110#special2 for more on the lawsuit). The changes seemed to really turn off some hard-core fans who crave ultimate control over the interface.

As I alluded to above, though, and will discuss in more detail later, there's been an updater since then that addresses this concern. Sadly, it may take time for many of those disappointed early users to give it another chance.

Frankly, I've never seen what the fuss was about, but then again I never felt compelled to customize the Studio interface. And after applying the HomeSite+ updater, this should be even less of an issue anyway.

To me, every CF Studio user who doesn't have Studio 5 should install HomeSite+. And with the updater to HomeSite + 5.2, even Studio 5 users might find it worth the tradeoff for any minor changes in the interface due to the lawsuit. As I'll discuss, Macromedia has also made the upgrade cost quite reasonable.

How to Get HomeSite+?

The important thing is that it's available *free* to those who own Dreamweaver MX or the fuller Studio MX package. It's actually on your Dreamweaver MX CD. Many don't know it's available because it's not on the install menu that's shown for that CD.

You simply need to browse the CD to find and execute the available "HomeSite+ Installer.exe" in the HomeSite+ directory. It will install just as CF Studio would.

Something to note is that when you install HomeSite+, it will prompt for a serial number. While it doesn't make it clear, I found that it took the Dreamweaver MX serial number.

Of course, it's perfectly acceptable to have HomeSite+ installed alongside Dreamweaver MX. Indeed, the two are even designed to support simultaneous editing of templates. See the TechNote at www.macromedia.com/support/dreamweaver/programs/migrating_cfs_to_dwmx/migrating_cfs_to_dwmx12.html for details.

Sadly, HomeSite+ is not available as a trial download, nor can you even buy it on its own. It only comes on the Dreamweaver CD. It's worth pointing out, however, that Dreamweaver MX currently sells for only $399, so for less than the previous cost of CF Studio 5, you get both products. An upgrade from Studio 4.5 or 5 to Dreamweaver MX is only $199. If you're a Studio 4.5 user, this combination of updating to HomeSite+ (which brings you all the features of Studio 5) and Dreamweaver MX is a compelling option. Studio 5 users get the addition of Dreamweaver MX plus the benefits of the HomeSite+ updater.

(To avoid confusion, it may be worth noting that you can buy HomeSite 5, but not HomeSite+. HomeSite+ adds the "Studio" features we all know and love to the base HomeSite 5 product, such as the database tab, the integrated debugger, remote development support, and more.) You can find out more about HomeSite+, especially in the context of HomeSite 5 and Dreamweaver MX, at www.macromedia.com/software/homesite/productinfo/faq/dw_hs_faq.html.

A Little About RDS and Development Mappings

Another facet to be aware of is that on a fresh installation of HomeSite+ (versus an upgrade), it will walk through creation of an RDS connection and Development Mappings.

The good news is that this approach helps ensure that you implement these important configuration options. The bad news is that if you're new to RDS and Development Mappings, or had ignored them in Studio 4/5, you'll possibly find all these prompts quite challenging.

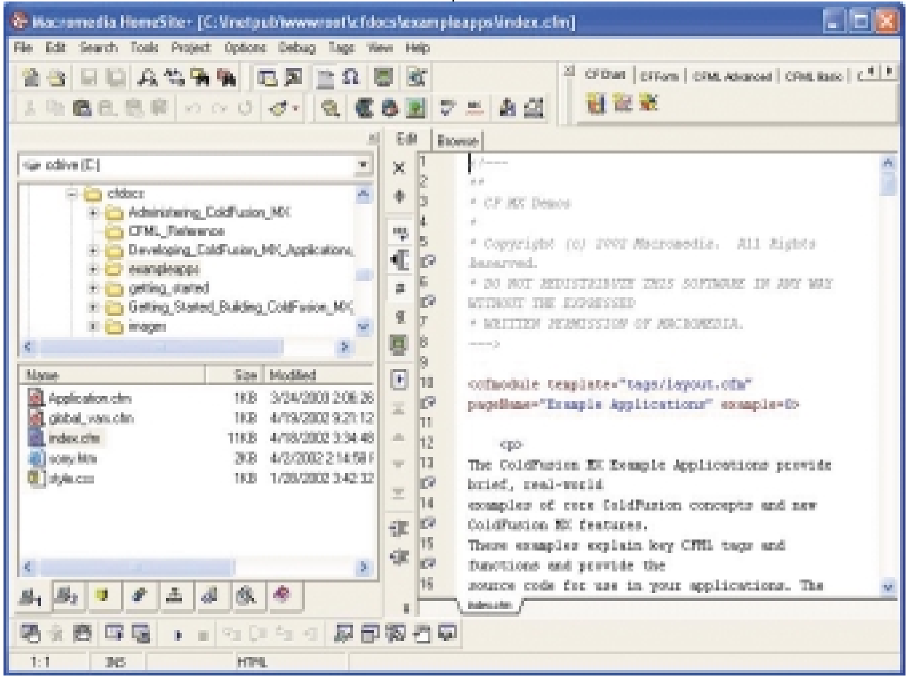
It's a shame, because getting RDS and development mappings set up correctly is very important to making the most out of HomeSite+. The database, deployment, and internal debugging tools rely on RDS while the internal and external browse features (and the debugging tool) rely on proper setting of the Development Mappings.

At least be aware that if you're setting it up to run with CFMX, and you've installed CFMX with the built-in Web server, you'll want to change the port for connecting to CFMX to 8500 (both in the configuration of the RDS server connection and in the Development Mapping where you indicate the URL for browsing your CF templates), as that's the default port that you use to browse pages with the CFMX built-in Web server.

When it requests that you enter a password when setting up an RDS connection, the one it's requesting is that defined in the CF Administrator for RDS security. (If creating an RDS server connection for your own localhost installation of CF, this password is the one you provided for this purpose at the time you installed CF.)

Note that you should *not* enter a value for "username" in that interface prompt. Just leave it empty. You should only enter a username and password in a rare case: if you're using CF 4 or 5 and your server is configured to use Advanced Security with the "ColdFusion Studio Authentication" feature. In that case you'd enter your own username and password as indicated by how that Advanced Security was configured in the CF Admin. This is not likely as it's a very rarely used, though powerful, feature. See more about this feature in the CF5 manual, "Advanced ColdFusion Administration," Chapter 5, and specifically, http://livedocs.macromedia.com/cf50docs/Advanced_ColdFusion_Administration/AdvSecurity11.jsp#1099137.

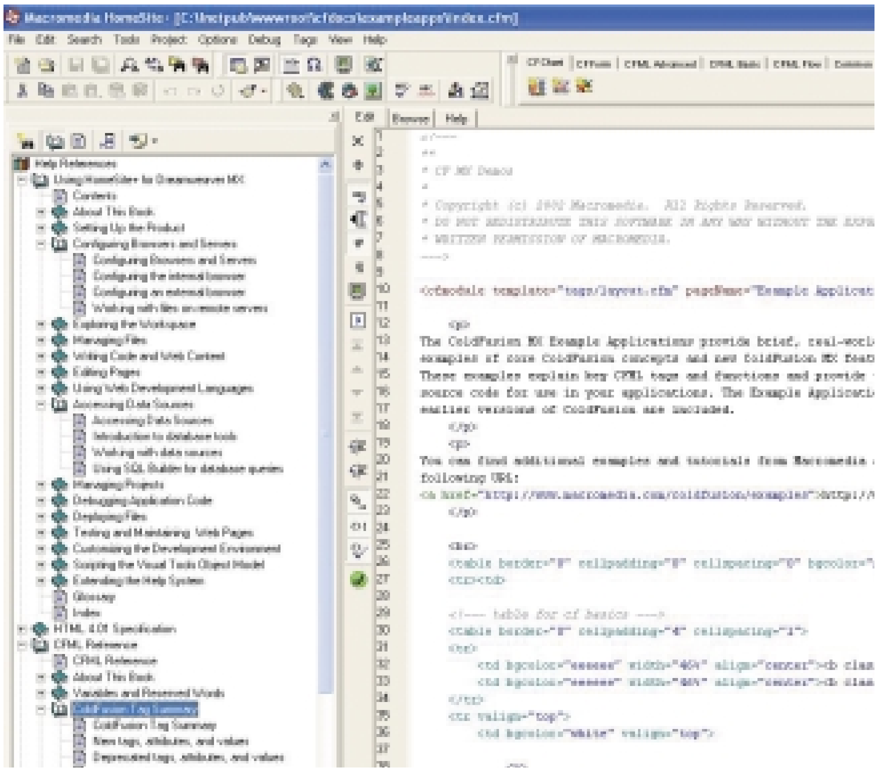
I'll have to defer further discussion of RDS and Development Mappings to another article, but I will point out a cou-


Macromedia HomeSite+ [C:\Inetpub\wwwroot\cfdocs\exampleapps\index.cfm]
File
Edit
Search
Tools
Project
Options
Debug
Tags
View
Help
CFChart
CFForm
CFML Advanced
CFML Basic
Edit
Browse
cdrive [C:]
cfdocs
Administering_ColdFusion_MX
CFML_Reference
Developing_ColdFusion_MX_Applications
exampleapps
getting_started
Getting_Started_Building_ColdFusion_MX
images
1:1
INS
HTML

Name	Size	Modified
Application.cfm	1KB	3/24/2003 2:06:26
global_vars.cfm	1KB	4/19/2002 9:21:12
index.cfm	11KB	4/18/2002 3:34:48
sorry.htm	2KB	4/2/2002 2:14:58 F
style.css	1KB	1/28/2002 3:42:12

```
<!---
**
* CF MX Demos
*
* Copyright (c) 2002 Macromedia.  All Rights
Reserved.
* DO NOT REDISTRIBUTE THIS SOFTWARE IN ANY WAY
WITHOUT THE EXPRESSED
* WRITTEN PERMISSION OF MACROMEDIA.
--->

<cfmodule template="tags/layout.cfm"
pageName="Example Applications" example=0>

    <p>
The ColdFusion MX Example Applications provide
brief, real-world
examples of core ColdFusion concepts and new
ColdFusion MX features.
These examples explain key CFML tags and
functions and provide the
source code for use in your applications. The
```

Figure 1: The CF Studio-like HomeSite+ interface



Macromedia HomeSite+ - [C:\Inetpub\wwwroot\cfdocs\exampleapps\index.cfm]

File Edit Search Tools Project Options Debug Tags View Help

CF Chart | CFForm | CFML Advanced | CFML Basic | CFML Flow | Common

Edit | Browse | Help

- Help References
  - Using HomeSite+ for Dreamweaver MX
    - Contents
    - About This Book
    - Setting Up the Product
    - Configuring Browsers and Servers
      - Configuring Browsers and Servers
      - Configuring the internal browser
      - Configuring an external browser
      - Working with files on remote servers
    - Exploring the Workspace
    - Managing Files
    - Writing Code and Web Content
    - Editing Pages
    - Using Web Development Languages
    - Accessing Data Sources
      - Accessing Data Sources
      - Introduction to database tools
      - Working with data sources
      - Using SQL Builder for database queries
    - Managing Projects
    - Debugging Application Code
    - Deploying Files
    - Testing and Maintaining Web Pages
    - Customizing the Development Environment
    - Scripting the Visual Tools Object Model
    - Extending the Help System
    - Glossary
    - Index
  - HTML 4.01 Specification
  - CFML Reference
    - CFML Reference
    - About This Book
    - Variables and Reserved Words
    - ColdFusion Tag Summary
      - ColdFusion Tag Summary
      - New tags, attributes, and values
      - Deprecated tags, attributes, and values

```
<!---
**
* CF MX Demos
*
* Copyright (c) 2002 Macromedia.  All Rights Reserved.
* DO NOT REDISTRIBUTE THIS SOFTWARE IN ANY WAY WITHOUT THE EXPR
* WRITTEN PERMISSION OF MACROMEDIA.
--->

<cfmodule template="tags/layout.cfm" pageName="Example Applicat

	<p>
The ColdFusion MX Example Applications provide brief, real-worl
examples of core ColdFusion concepts and new ColdFusion MX feat
These examples explain key CFML tags and functions and provide
source code for use in your applications. The Example Applicati
earlier versions of ColdFusion are included.
	</p>
	<p>
You can find additional examples and tutorials from Macromedia
following URL:
<a href="http://www.macromedia.com/coldfusion/examples">http://
	</p>

	<br>
	<table border="0" cellpadding="0" cellspacing="0" bgcolor="
	<tr><td>

	<!--- table for cf basics --->
	<table border="0" cellpadding="4" cellspacing="1">
	<tr>
		<td bgcolor="eeeeee" width="40%" align="center"><b clas
		<td bgcolor="eeeeee" width="60%" align="center"><b clas
	</tr>
	<tr valign="top">
		<td bgcolor="white" valign="top">
```

Figure 2: Help references window in HomeSite+

ple of resources in the meantime. First, there is a "Using HomeSite+" manual in the Help>Open Help References Window (see Figure 2).

Of course, you won't see that help until you complete the installation. But after completing the installation and setup, see Chapter 2 of that manual, especially the last section about setting up RDS and Development Mappings and the previous sections on configuring the internal and external browsers. And see Chapter 8 about using the database tab. Note that I've expanded these chapters in the help toolbar shown in Figure 2.

Again, I'll include more on that in a future article, but in the meantime I also discuss RDS configuration and some security and other issues in a new User Group presentation I'm offering, called "RDS: from Stress to Success." To see the presentation slides, see www.systemanage.com/presentations. You'll see that I've also offered other talks on working with Studio over the years. Check them out.

Update to HomeSite+ Available

You may have heard that there was an updater for ColdFusion MX (I wrote about it in last month's **Journeyman** article, "CFMX Updaters 1-2-3," ***CFDJ***, Vol. 5, issue 5 www.sys-con.com/coldfusion/article.cfm?id=600). If you didn't know it, there's also an updater for Dreamweaver MX (www.macromedia.com/support/dreamweaver/ts/documents/updater.htm).

Many of you will be surprised to learn that there is an updater for HomeSite+ (which updates it from the base 5.1 to a new 5.2 version). It's at www.macromedia.com/cfusion/resourcecenter/rc_driver.cfm?pageName=hsp%5Fupdater, which is behind a login/registration page just as all MM downloads are now.

As I mentioned earlier, one of the main things it solves is offering more flexibility in customizing the toolbars and resource windows. See the Release Notes at www.macromedia.com/support/homesite/releasenotes/plus/releasenotes_plus_52.html, which includes a long list of what's been fixed (see my favorite, listed as a fix to bug 28406, which long plagued Studio users using the Extended Find feature).

Adding CFML Reference to the HomeSite+ (and CF Studio) Help Tab

One thing the updater doesn't fix is that HomeSite+ does not by default include the CFMX version of the CFML Reference manual in the Help Reference Tab (Help>Help References Window). This is a curious oversight.

It's worth clarifying that the HomeSite+ "tag help" feature does indeed include help for CFMX tags and functions. If you place the cursor on one and press F1, you are shown help for that tag or function, because HomeSite+ does indeed include tag help, tag editors, tag insight, etc. for CFMX. But if you use the help tab in the resource toolbar to see the available manuals, the CFML reference just isn't there by default (indeed, none of the CF manuals are there).

Macromedia has solved the problem, at least a little. You can go to http://download.macromedia.com/pub/homesite/updates/cfml_ref.zip and download a zip file that includes the directory needed to provide that CFMX version of the CFML Reference Manual within the help mechanism. Just follow the instructions in the CFMLRef_install.txt file provided in the zip file telling where to install the directo-

HomeSite 5 manuals not the same as HomeSite+

While we're on the subject of manuals, in addition to the "Using HomeSite+" manual offered in the Help References window, you may also notice that you can find the "Using HomeSite" manual in HTML form at the livedocs.macromedia.com site and in PDF form at www.macromedia.com/v1/documents/hs50/homesite5.pdf. Take note, however, that these are not the same as what is in the Help References Window of HomeSite+. These show the manual for HomeSite 5, not the HomeSite+. There are a few chapters missing for features such as the database tab, the debugger, and more that are not in the HomeSite 5 product. Stick with the manual as offered in the HomeSite+ help. Or if you really want an HTML or PDF version for easier access, look at the "Using CF Studio 5" manual at livedocs.macromedia.com or www.macromedia.com/v1/documents/cf50/cfstudio5.pdf. Things haven't changed enough to cause problems reading that book.

ry of files within the HomeSite+ installation directory. It also tells you how you can improve the appearance of the manual topics inside the help tab once installed (ordering them by chapter) by editing the booktree.xml file, as explained.

In Figure 2, you can see that I have the CFML reference manual installed. Sadly, the fix only offers the CFML reference, not the other manuals in the CFMX documentation set. Check out http://livedocs.macromedia.com for an online set of the docs, or if you installed CFMX on your workstation, the docs are optionally installed with CFMX.

And here's great news for Studio 4.5/5 users: while the instructions apply to HomeSite+, the great news is that they're equally applicable to previous releases of HomeSite and Studio. Yep, you can add the CFMX Reference Manual to the Studio 4.5 or 5 help tab (Help>Open Help References Window).

Indeed, on a related but different matter, I'll point out that for both HomeSite+ and CF Studio, you can add other references (SQL, JavaScript, and more) to the Help References window. See my January 2002 ***CFDJ*** article, "Adding New Help Topics to Studio," at www.sys-con.com/coldfusion/articlea.cfm?id=392.

Adding CFMX Tag/Function Support to Studio 4.5/5

If you're sticking with Studio 4.5 or 5, you may be thinking that while it's nice to be able to add the CFMX version of the CFML Reference Manual to the help tab, what you'd really like is to get the CFMX tags and functions recognized in the tag editor, tag insight, tag completion, tag help, and other such features.

There's great news for you, too. Macromedia has offered a "tag updater" for CF Studio 4.5 and 5. See www.macromedia.com/software/coldfusionstudio/productinfo/resources/tag_updaters.

Conclusion

So, now you have quite a bit of information about using HomeSite+, including how to upgrade it, install it, extend it, and a little about configuring it. I hope you'll seriously consider it and explore it further.

You may want to check out the available HomeSite tips area at www.macromedia.com/cfusion/tipsubmission/maintopic_browse.cfm?topicId=12. But more important, be sure to check out the manual I referred to, "Using HomeSite+," offered in the Help References tab. Many people never read that manual and just don't know what they're missing. I'll talk more about some of these things in future articles. I'm glad to see that we Studio fans still have our favorite editor available. It may have a new name, but for many of us, it's still the best editor out there.

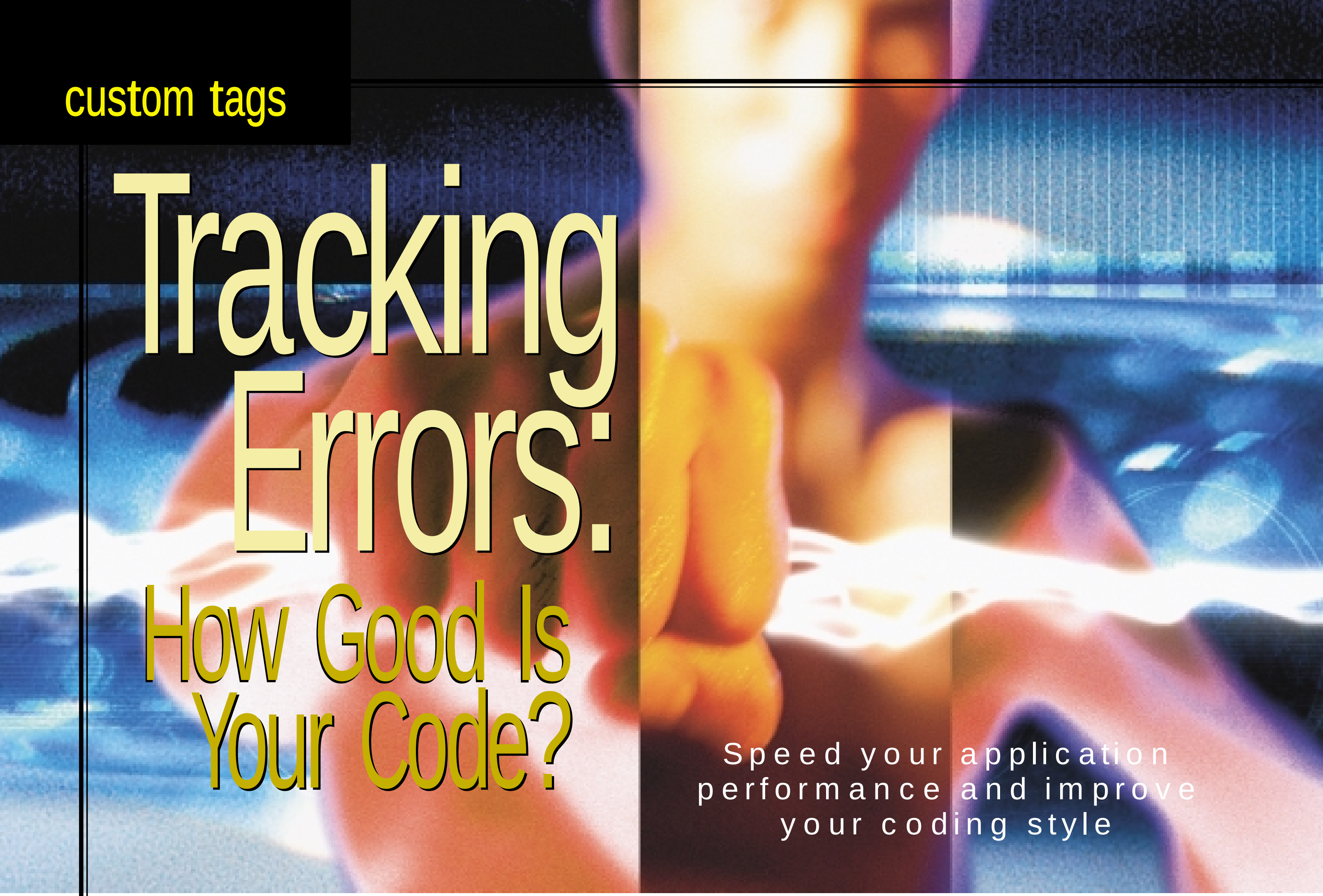
About the Author

Charlie Arehart is co-technical editor of **ColdFusion Developer's Journal** *and a Macromedia Certified Advanced ColdFusion developer and trainer. He has recently become CTO of New Atlanta Communications, makers of BlueDragon. In his new role, he will continue to support the CFML community, contributing to several CF resources, and speaking frequently at user groups throughout the country.*

charlie@newatlanta.com

PAPERTHIN

www.paperthin.com



Tracking Errors: How Good Is Your Code?

Speed your application performance and improve your coding style

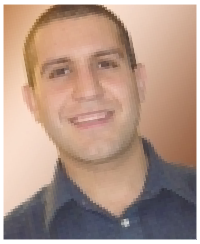
Nobody likes errors, but the bottom line is, they're a fact of programming life. The better you are at pinpointing them, the better your code will get. Unfortunately, ColdFusion has never provided very good functionality for tracking the errors that occur within your production applications. I mean how many of you actually look through your application.log? For those of us with busy servers and busy schedules, that becomes an impossibility...

Thus good tools are needed. ColdFusion 4 and 5 and MX include a Log Files reporting tool in the CF Administrator. For some, that still may not be enough. It would be helpful, for instance, to extract the log of errors into a database for subsequent reporting. Right off the bat, it seems Allaire/Macromedia didn't fully think things through as the application.log is not easily parsed.

The problem is, when certain types of errors occur (such as ODBC errors), quotes appear within the text of the error and that interferes with the quotes used as field delimiters for the log. Rather than give up on our goal, however, we can find an alternative solution. What if we could handle errors ourselves rather than rely solely on after-the-fact log analysis?

Enter ColdFusion's error handling. Since version 4.5, Macromedia has made available improved error handling. One of the early restrictions CF developers faced was not being able to run any ColdFusion code on an error-handling page. It was the old chicken and egg problem – what happens if an error occurs on the error-handling page? Fortunately we don't have to worry about those kinds of restrictions anymore.

There are several ways of dealing with errors by executing ColdFusion tags, such as CFTRY/CFCATCH, the CFERROR tag (using TYPE=Exception), and the site-wide error handler that can be enabled through the CF Administrator. These have been discussed in various previous ***CFDJ*** articles, such as Charlie Arehart's four-part series starting in October 2000 at www.sys-con.com/coldfusion/article.cfm?id=165.



By Joe Danziger

In this article, we'll take advantage of the site-wide error handler, which will catch any errors that fall through the cracks and interrupt a page from running.

There are a couple of things to be aware of about error handling, such as with the site-wide handler. First, when using it to intercept an error, CF will not write the error to the application.log. You'll have to do that yourself if you really want to (see Listing 1).

Listing 1: Writing out to application.log

```
<cflog text="#error.diagnostics#" log="APPLICATION" type="Error"
thread="yes" date="yes" time="yes" application="yes">
```

Since our end result was to parse the application.log and insert that into a database, we'll skip the step of writing out the log and insert each error directly into the database as it occurs.

Listing 2 contains the database tables that we'll use for our application. Note the "fixID" field at the end of the "errorLog" table. We'll set that as we correct our errors so that we know which ones have been fixed.

Listing 2: Table and index creation

```
CREATE TABLE errorLog (
    logID          int IDENTITY (1,1) NOT NULL,
    application    varchar(20)         NULL,
    errorDetail    varchar(2200)       NULL,
    errorBase64    varchar(3000)       NULL,
    errorStamp     datetime   NOT NULL,
    errorType      varchar(30)         NULL,
    browser        varchar(100)        NULL,
    remoteAddr     varchar(15)         NULL,
    httpReferer    varchar(255)        NULL,
    server         varchar(30)         NULL,
    template       varchar(255)        NULL,
    scriptName     varchar(200)        NULL,
    queryString    varchar(120)        NULL,
    fixID          smallint   NULL
)

PRIMARY KEY: logID
INDEXES: fixID, errorDetail, errorBase64, errorType, application, tem-
plate

CREATE TABLE errorFixes (
    fixID          int IDENTITY (1,1) NOT NULL,
    logID          int                 NULL,
    fixDetail      varchar(500)        NULL,
    fixStamp       datetime   NULL
)

PRIMARY KEY: fixID
INDEXES: logID
```

Listing 3 contains the CF code for your site-wide error handler that will insert the error into the database. There are several error types that are common to ColdFusion and might be present in your application (see Table 1). We'll later produce a report that groups things by these types, so we'll add some code right before the insert query to determine the type of error that has occurred. Feel free to add additional code to catch specific errors that you want grouped into the "Other" category.

Database error
Request timeout
Session timeout
Error resolving parameter
Error evaluating expression
Compilation error
Other errors (LDAP, File, etc..)

Table 1: Error types

Listing 3: Inserting the error via SQL

```
<Cflock scope="application" type="readonly" timeout="10">
<Cfset variables.appName = #application.applicationName#>
</cflock>

<cfif Left(trim(error.diagnostics),4) is 'ODBC' or Left(trim(error.diag
    nostics),5) is 'OLEDB' or Left(trim(error.diagnostics),30) is 'Error
    Executing Database Query'>
        <cfset errorType = "Database Error">
<cfelseif Left(trim(error.diagnostics),17) is 'Request timed out'>
    <cfset errorType = "Request Timeout">
<cfelseif error.diagnostics contains 'Error resolving parameter'>
    <cfset errorType = "Error Resolving Parameter">
<cfelseif error.diagnostics contains 'Error resolving parameter <B>SESSION.'>
    <cfset errorType = "Session Timeout">
<cfelseif Left(trim(error.diagnostics),53) is '<P>An error occurred while
    evaluating the expression:'>
    <cfset errorType = "Error Evaluating Expression">
<cfelseif Left(trim(error.diagnostics),30) is 'Just in time compilation error'>
    <cfset errorType = "Compilation Error">
<cfelse>
    <cfset errorType = "Other Error">
</cfif>

<cfquery name="InsertLogRecord" datasource="#request.datasource#">
    INSERT INTO errorLog (errorStamp, errorDetail, errorBase64, errorType,
    template, queryString, browser, remoteAddr, httpReferer, application,
    server, scriptName) VALUES (#CreateODBCDateTime(error.DateTime)#,
    '#left(trim(error.diagnostics),2200)#',
    '#Trim(Left(ToBase64(error.diagnostics),3000))#', '#errorType#',
    '#error.template#', '#error.queryString#','#error.browser#',
    '#error.RemoteAddress#', '#ERROR.HTTPReferer#', '#variables.appName#',
    '#cgi.server_name#', '#cgi.script_name#')
</cfquery>
```

Note that you could add more to this logging of information, such as writing out session identifiers (CFID, CFTOKEN, and/or JSessionID if using J2EE sessions in CFMX). Also, the locking of access to the application.applicationname variable is something that could be removed if this code is run in CFMX.

Of course, this listing is just a fragment of what you could do in the error handler. For more information on site-wide error handlers, including how to create one, how to provide notification to the user about the error, and optionally how to send e-mail notification to someone in your shop about the error, see the second part in the previously mentioned series, "Toward Better Error Handling – Part 2: Site Wide Error Handling" from December 2000, at www.sys-con.com/coldfusion/article.cfm?id=181.

You may notice that I chose to create a column called errorBase64. Let me explain, and you may need to make some adjustments based upon your database. Specifically, if your database cannot handle sub-selects (as we'll explain later and see in Listing 5), we cannot do comparisons directly within the database given the approach we'll be using. Therefore we'll need a way to encode special characters such as quotes. Most enterprise databases such as Oracle and MS SQL will have no problems with sub-selects, and MS Access supports them as well, but some databases such as mySQL do not (although support is coming soon).

Fortunately, there is an easy workaround. You can add one additional field to your database table to store the error encoded in

Base64 (we've reflected this in the code listings). Those of you who store binary files directly in your databases may already be familiar with the ToBase64() function which does just that. If your database does handle sub-selects, you can ignore all references to Base64 as you won't need to do any conversions.

Now that we're collecting all of our errors, we need a way to make sense of it all.

So what's next? Well, we need a way to sort and organize our errors so that we can start to see which are our biggest problems. In our completed application, we've created a CFC called error.cfc with a number of CFFUNCTION methods that are used to search the database we've created. There are several ways you may want to sort including by template, by application, newest first, most frequent, and by type of error. Later we'll look at some different ways to sort and view your errors. For now, we'll just list the newest errors first (see Listing 4). Notice that we added the "WHERE fixID IS NULL" line to our query in order to not include errors that have already been corrected.

Listing 4: Newest errors

```
<cffunction name="byNewest" displayName="Newest Errors" access="public"
   returnType="query" output="false">
      <cfset var getErrors = "">
   <cfquery name="getErrors" datasource="#request.datasource#">
         SELECT logID, application, errorDetail, errorStamp FROM
         errorLog
                 WHERE fixID IS NULL
                         ORDER BY errorStamp DESC
   </cfquery>
   <cfreturn getErrors>
</cffunction>
```

We might then call that CFC using code such as:

```
<cfinvoke component="errors" method="byNewest"
   returnvariable="GetErrors"></cfinvoke>
<cfoutput query="getErrors">#application# : #errorDetail# :
   #errorStamp#<br></cfoutput>
```

The code in Listing 5 will display an individual error. You should set up a separate page to drill down to this info. The GetSimilar query will show you how many times the specific error you're viewing appears in the log. You'll need to comment out either the "with sub-selects" or "without sub-selects" portion of the code depending upon your database's capabilities. We then add one column to the end of our original query listing the "SimCount," or the number of times a matching error appears.

Listing 5: Get detail

```
<cffunction name="getDetail" displayName="Get Error Detail"
         access="public" returnType="query" output="false">
   <cfargument name="logID" type="numeric" required="true">
   <cfset var getError = "">
   <cfset var getSimilar = "">
   <cfset var simCount = arrayNew(1)>
   <cfquery name="getError" datasource="#request.datasource#">
         SELECT * FROM errorLog WHERE logID = #ARGUMENTS.logID#
   </cfquery>

      <!--- with sub-selects --->
   <cfquery name="GetSimilar" datasource="#request.datasource#">
         SELECT count(*) as numSimilar FROM errorLog
                 WHERE errorDetail = (SELECT errorDetail FROM
                 errorLog WHERE logID = #ARGUMENTS.logID#)
                         AND logID != #ARGUMENTS.logID#
   </cfquery>

      <!--- without sub-selects --->
   <cfquery name="GetSimilar" datasource="#request.datasource#">
         SELECT count(*) as numSimilar FROM errorLog
         WHERE errorBase64 = '#getError.errorBase64#'
         AND logID != #ARGUMENTS.logID#
   </cfquery>

   <!--- create an array and add column to query --->
   <cfset simCount[1] = GetSimilar.numSimilar>
   <cfset QueryAddColumn(getError, "SimCount", simCount)>
   <cfreturn getError>
</cffunction>
```

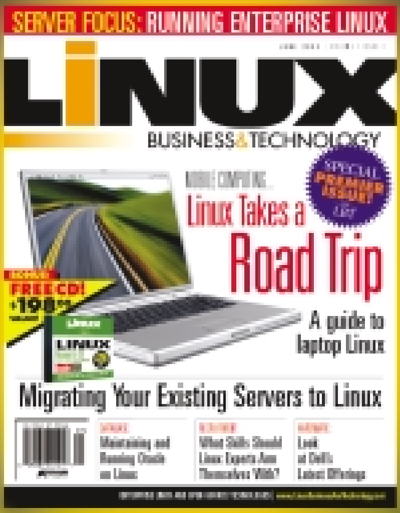
To keep our error log manageable, we need a way to mark errors as corrected once the code has been fixed. At the bottom of the screen displaying an individual error, we'll add a textarea field for entering the fix we performed. When we submit a fix, we'll mark the error as solved (see Listing 6) and also update all of the matching errors since those should now be corrected also (note the slight query changes depending on whether your database can handle sub-selects).

Listing 6: Enter problem fix

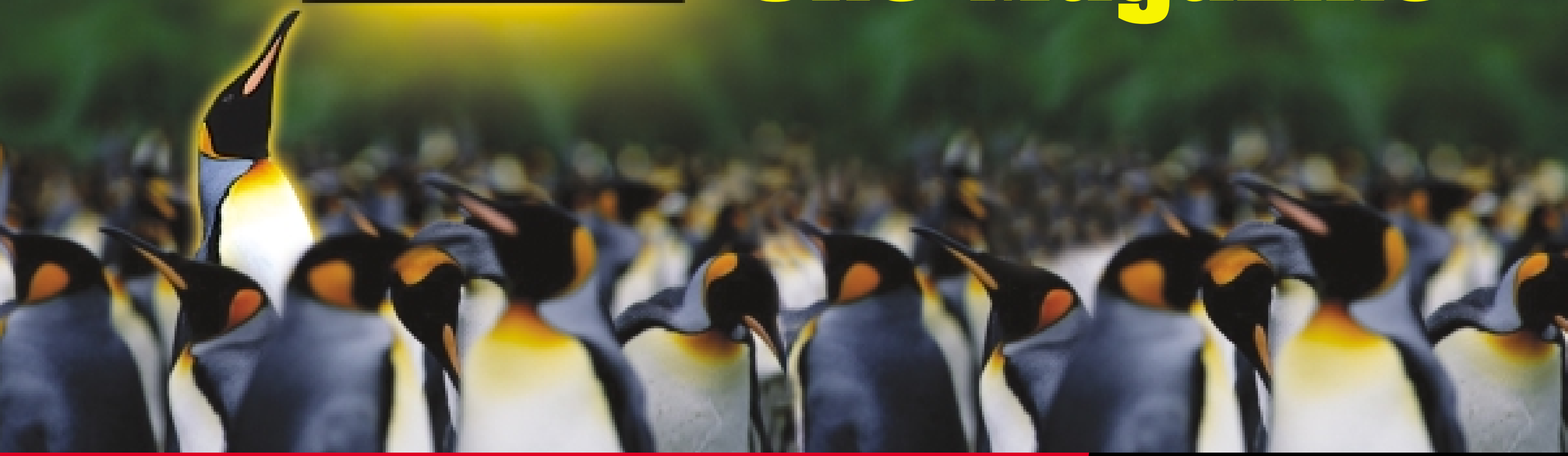
```
<cffunction name="enterFix" displayName="Enter Problem Fix"
  access="public" returnType="void" output="false">
   <cfargument name="logID" type="numeric" required="true">
   <cfargument name="errorFix" type="string" required="true">
   <cfset var enterFix = "">
   <cfset var getMatching = "">
   <cfset var getBase64 = "">
   <cfset var getMatching = "">
   <cfset var updateLogTable = "">
   <cfquery name="enterFix" datasource="#request.datasource#">
         INSERT INTO errorFixes (logID, fixDetail, fixStamp)
         VALUES (#ARGUMENTS.logID#, '#ARGUMENTS.errorFix#',
         #CreateODBCDateTime(Now())#)
   </cfquery>

      <!--- with sub-selects --->
   <cfquery name="GetMatching" datasource="#request.datasource#">
         SELECT logID FROM errorLog WHERE errorDetail =
                 (SELECT errorDetail FROM errorLog WHERE logID =
#ARGUMENTS.logID#)
   </cfquery>

      <!--- without sub-selects --->
   <cfquery name="GetBase64" datasource="#request.datasource#">
              SELECT errorBase64 FROM errorLog WHERE logID = #ARGU
                 MENTS.logID#
   </cfquery>
```

Premiering June 2003 at JavaOne

www.sys-con.com


SERVER FOCUS: RUNNING ENTERPRISE LINUX
LINUX BUSINESS & TECHNOLOGY
SPECIAL PREMIER ISSUE!
Linux Takes a Road Trip
A guide to laptop Linux
FREE CD!
Migrating Your Existing Servers to Linux

Millions of Linux Users One Magazine



Linux Business & Technology

There is no escaping the penetration of Linux into the corporate world. Traditional models are being turned on their head as the open-for-everyone Linux bandwagon rolls forward.

Linux is an operating system that is traditionally held in the highest esteem by the hardcore or geek developers of the world. With its roots firmly seeded in the open-source model, Linux is very much born from the "if it's broke, then fix it yourself" attitude.

Major corporations including IBM, Oracle, Sun, and Dell have all committed significant resources and money to ensure their strategy for the future involves Linux. Linux has arrived at the boardroom.

Yet until now, no title has existed that explicitly addresses this new hunger for information from the corporate arena. *Linux Business & Technology* is aimed squarely at providing this group with the knowledge and background that will allow them to make decisions to utilize the Linux operating system.

Look for all the strategic information required to better inform the community on how powerful an alternative Linux can be. *Linux Business & Technology* will not feature low-level code snippets but will focus instead on the higher logistical level, providing advice on hardware, to software, through to the recruiting of trained personnel required to successfully deploy a Linux-based solution. Each month will see a different focus, allowing a detailed analysis of all the components that make up the greater Linux landscape.

Regular features will include:

- *Advice on Linux Infrastructure*
- *Detailed Software Reviews*
- *Migration Advice*
- *Hardware Advice*
- *CEO Guest Editorials*
- *Recruiting/Certification Advice*
- *Latest News That Matters*
- *Case Studies*

SAVE 30% OFF!

REGULAR ANNUAL COVER PRICE $71.76

YOU PAY ONLY $49.99

12 ISSUES/YR

*OFFER SUBJECT TO CHANGE WITHOUT NOTICE

SUBSCRIBE TODAY!

WWW.SYS-CON.COM OR CALL 1-888-303-5282

LINUX EDGE conference&expo

June 3–5	LONDON
June 24–26	BERLIN
September	HONG KONG
October	CALIFORNIA

FOR ADVERTISING INFORMATION:

CALL 201 802.3020 OR VISIT WWW.SYS-CON.COM

SYS-CON MEDIA
The World's Leading i-Technology Publisher

```
    <cfquery name="GetMatching" datasource="#request.datasource#">
            SELECT logID FROM errorLog WHERE errorBase64 =
              '#GetBase64.errorBase64#'
  </cfquery>

  <cfquery name="updateLogTable" datasource="#request.datasource#">
       UPDATE errorLog SET fixID = #enterFix.id# WHERE logID IN
              (#ValueList(getMatching.logID)#)
  </cfquery>
</cffunction>
```

We now have a way to list, view, and fix the latest errors that occur. Next we'll look at various other ways of sorting and viewing our data. Previously we only listed the newest errors first – now we'll sort by application, template, top errors, and error types (see Listings 7–10). These views will allow you to see which errors are occurring the most based upon your selection criteria. The errCount alias in each query will keep track of the number of errors for that grouping and the maxStamp alias will hold the last date that an error for that grouping has occurred.

Listing 7: Errors by application

```
<cffunction name="byApplication" displayName="Errors By Application"
       access="public" returnType="query" output="false">
  <cfset var getErrors = "">
<cfquery name="getErrors" datasource="#request.datasource#">
       SELECT application, count(logID) as errCount,
              max(errorStamp) as maxStamp FROM errorLog
              WHERE fixID IS NULL
                     GROUP BY application
                     ORDER BY errCount DESC
</cfquery>
  <cfreturn getErrors>
</cffunction>
```

This could be called with:

```
<cfinvoke component="errors" method="byApplication"
  returnvariable="GetErrors"></cfinvoke>
<cfoutput query="getErrors">#application# : #errCount# :
  #maxStamp#<br></cfoutput>
```

Listing 8: Errors by template

```
<cffunction name="byTemplate" displayName="Errors By Template"
       access="public" returnType="query" output="false">
  <cfset var getErrors = "">
  <cfquery name="getErrors" datasource="#request.datasource#">
       SELECT template, application, count(*) as errCount,
              max(errorStamp) as maxStamp FROM errorLog
              WHERE fixID IS NULL
                     GROUP BY template, application
                     ORDER BY errCount DESC
  </cfquery>
  <cfreturn getErrors>
</cffunction>
```

This could be called with:

```
<cfinvoke component="errors" method="byTemplate"
  returnvariable="GetErrors"></cfinvoke>
<cfoutput query="getErrors">#template# : #application# : #errCount# :
  #maxStamp#<br></cfoutput>
```

Listing 9: Top errors

```
<cffunction name="byTopErrors" displayName="Top Errors" access="public"
returnType="query" output="false">
  <cfset var getErrors = "">
  <cfquery name="getErrors" datasource="#request.datasource#">
       SELECT errorDetail, application, count(logID) as errCount,
                     max(errorStamp) as maxStamp, max(logID)
                     as maxLogID
              FROM errorLog
              WHERE fixID IS NULL
                     GROUP BY errorDetail, application
                     ORDER BY errCount DESC
  </cfquery>
  <cfreturn getErrors>
</cffunction>
```

This could be called with:

```
<cfinvoke component="errors" method="byTopErrors"
  returnvariable="GetErrors"></cfinvoke>
<cfoutput query="getErrors"> #application# : #errorDetail# : #errCount#
  : #maxStamp#<br></cfoutput>
```

Listing 10: Errors by error type

```
<cffunction name="byErrorType" displayName="Errors By Error Type"
access="public" returnType="query" output="false">
  <cfset var getErrors = "">
  <cfquery name="getErrors" datasource="#request.datasource#">
       SELECT errorType, count(*) as errCount, max(errorStamp) as
              maxStamp FROM errorLog
              WHERE fixID IS NULL
                     GROUP BY errorType
                     ORDER BY errCount DESC
  </cfquery>
  <cfreturn getErrors>
</cffunction>
```

This could be called with:

```
  <cfinvoke component="errors" method="byErrorType"
returnvariable="GetErrors"></cfinvoke>
<cfoutput query="getErrors"> #errorType# : #errCount# :
#maxStamp#<br></cfoutput>
```

Used effectively, this tool will speed your application performance and improve your coding style. Good luck!

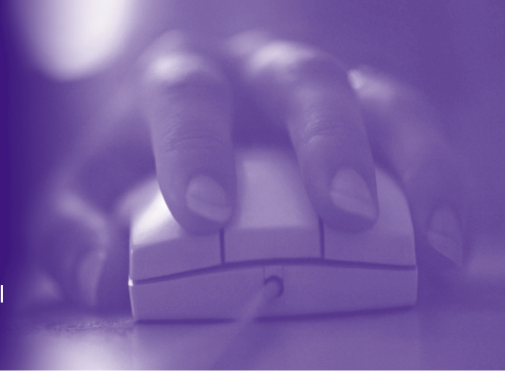
About the Author

Joe Danziger is the founder and president of DJCentral.com, an online promotional tool for disc jockeys and other members of the electronic dance music industry. He has been developing professional ColdFusion solutions for over six years since version 1.5.

joe@djcentral.com

SaturdaySessions

We understand the pressures of work and how difficult it can be to get time off. That is why we have designed this workshop to be held in one day and, as a special bonus, on the weekend, so no days off from work. **Your boss will be happy!**



JDJ Workshop with Alan Williamson

M	T	W	T	F	S	S

Coming to you...

April:	**May:**	**June:**
NEW YORK	BOSTON	ATLANTA
WASHINGTON, DC	TORONTO	RALEIGH

Performance > Efficiency > Reliability

This one-day intensive workshop is designed for developers who wish to increase the efficiency and reliability of their code development.

1) The day will begin by looking at the various hints and tips you can utilize at the code level to improve the quality and reduce the number of bugs you have to contend with.

2) The next part will look at Apache's Ant and how you can use this freely available tool for your own development, irrespective of your IDE.

3) Last, and most important, as the old saying goes: "You can never do enough testing." This session will look at JUnit and show you how to start building test harnesses for your code so you can begin your testing strategy.

>Performance

Java is a powerful language. While it offers a rich array of tools, the fundamentals mustn't be overlooked. Improving your code at the core layer will result in great improvements in efficiency and produce (hopefully) less bugs. We'll look at the do's and don'ts of programming and learn many hints and tips that will accelerate your Java coding.

>Efficiency with Ant

Apache's Ant is a powerful scripting tool that enables developers to define and execute routine software development tasks using the simplicity and extensibility of XML. Ant provides a comprehensive mechanism for managing software development projects, including compilation, deployment, testing, and execution. In addition, it is compatible with any IDE or operating system.

> Reliability with JUnit

A critical measure of the success of software is whether or not it executes properly. Equally important, however, is whether that software does what it was intended to do. JUnit is an open-source testing framework that provides a simple way for developers to define how their software should work. JUnit then provides test runners that process your intentions and verify that your code performs as intended. The result is software that not only works, but works in the correct way.

What you will receive...

- ✓ INTENSIVE ONE-DAY SESSION
- ✓ DETAILED COURSE NOTES AND EXCLUSIVE ONLINE RESOURCES
- ✓ JDJ CD ARCHIVE

ONLY $295*

*JDJ subscribers
($395 non-subscribers)

*GROUP DISCOUNTS AVAILABLE

To Register

www.sys-con.com/education

Call 201 802-3058

SPONSORED BY JAVA DEVELOPER'S JOURNAL

PRODUCED BY SYS-CON EVENTS

PRESENTED BY i-TECHNOLOGY EDUCATION

Design Patterns in ColdFusion: Composite Pattern

Aggregating CFC objects with polymorphism PART 4 OF A SERIES

As ColdFusion developers, we tend to build applications based on pages that display information and queries to retrieve information and save information. We tend to think solely of how we are going to implement the code required to perform a certain function. Many times we are not able to consider the commonality of a certain problem and whether an already established design exists somewhere.

Often we are not able to set aside time to do this research. Knowing and utilizing design patterns allows us to think in a common language, with an ever-expanding library of well thought out solutions to consider when architecting an application. Recognition of these patterns in our everyday jobs can make us significantly more productive and seem even smarter than we are.

By Brendan O'Hara

In the past three issues of *CFDJ*, I have been showing you how other languages use object-oriented design patterns and how those patterns can be implemented in ColdFusion MX. This month's pattern differs significantly from the previous three so we should probably go over what makes it different. Let's start by examining the three main categories that design patterns fall into: creational, structural, and behavioral.

- ***Creational patterns:*** Encapsulate and abstract the instantiation process. That is, we may make creating a CFC through <cfobject> or createObject() the responsibility of another CFC or of a custom tag. It can be useful for simplifying the code needed to initiate certain processes. Since ColdFusion MX is not purely object oriented, there is significantly less "object creation." This means there is probably less insight to be gained reviewing creational patterns. However, the Singleton pattern comes to mind as a creational pattern that can relate to managing the state of an application.
- ***Structural patterns:*** Deal with how inheritance, interface implementation, and most specifically object composition make objects more suitable for a wide array of tasks. In other words, a CFC that contains other CFC instances can utilize these instances to perform operations within its methods. This can allow us to combine related CFCs and operations into a single interface.
- ***Behavioral patterns:*** Encapsulate varying algorithms and control flow, assigning responsibilities and fostering communication between objects when appropriate. The Template Method pattern, Iterator pattern, and Strategy pattern, which have been covered in previous issues, are all behavioral patterns. In one way or another they abstract the behavior of a CFC's methods, allowing us to implement multiple behaviors from a single interface.

One piece of commonality among these three is that they often deal with encapsulating not only common code but varying code, although for substantially different reasons. Additionally, because we are advised in object-oriented design to "program to an interface, not an implementation," they usually provide common interfaces. This month we will take a look at our first structural design pattern, the Composite pattern.

The Composite Pattern

As we do every month when talking about object-oriented design patterns, we like to refer to the original "intent" of the pattern as laid out in the book, *Design Patterns: Elements of Reusable Object-Oriented Software*. As the preeminent work on the subject of design patterns, its authors, the so-called "Gang of Four," established the general intent of the Composite pattern to be the following:

Compose objects into tree structures to represent whole-part hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

The Composite pattern, and more specifically composition by aggregation, helps us to create tree structures to encapsulate not just related objects but a series of related objects that, themselves, may contain other objects of the same or related types. This results in a sort of object family tree.

Now all we need is to be able to build an interface that allows us to treat parent and child objects uniformly. We will use a fairly standard example of composition by aggregation using employees, bosses, and subordinates to explore what structural patterns can help us achieve. But before we get too deep let's quickly run down what an object is and how it's implemented in ColdFusion. Although a review for some of you, it's important because the object plays such a vital role in understanding design patterns.

What Is an Object?

This is actually a pretty simple question with a fairly complicated answer. The word *object* itself is used in several contexts in the world of OOP that tends to make things somewhat confusing. We call an object such because it models objects from real life including their state, attributes, and behaviors. ColdFusion has never been a purely object-oriented language but with the release of ColdFusion MX it has a much more object-friendly construct known as ColdFusion Components or CFCs.

A simple example of an object is a person. We may want to know if the person is sleeping, awake, tired, or hungry. These are examples of the person's *state*. We may want to know the person's height or hair color. These are examples of *attributes*. State and attributes are implemented as properties of a CFC object. In addition, a person can perform an action such as walk, run, or eat. These are examples of *behavior*.

Behaviors are implemented as methods in a CFC object. In addition, normally in OOP you would create "getter" and/or "setter" methods (i.e., getName() or setName("Rob Ray")) for each property, which is allowed to be set or retrieved manually. You may also create a GetPerson() method, which returns a query object with that person's properties after querying them from a database. Simplified code for a person object is listed below:

```
<cfcomponent displayname="Person">
   <cfparam name="My" type="struct"
    default="#structNew()#">
   <cfset My.Name = "">
   <cffunction name="init" access="public"
   returntype="struct">
         <cfargument name="Name"
          required="Yes" default="">
   <cfset My.Name = Arguments.Name>
   </cffunction>
   <cffunction name="display" access="public"
    returntype="void" output="Yes">
         - #My.Name#<br>
   </cffunction>
   <cfreturn this>
</cfcomponent>
```

You will notice that on my "init" methods I return "this". As you may or may not have guessed from the ReturnType "this" returns a pointer to the currently instantiated CFC. I find this helpful for method chaining, which is demonstrated below.

Here is how we instantiate a person CFC object:

```
<cfset MyPerson = createObject("component",
   "Person").init("Bob Smith")>
```

First, the CreateObject function returns the current CFC instance, which then invokes the init() method, which also returns the current CFC instance. The variable MyPerson actually receives the instance returned by the last method in the chain which in this case is init(). Calling the init() method immediately after the createObject() call is the example of method chaining I was referring to. The following is the alternative syntax for calling a "setter" method or any method that doesn't return a value.

```
<cfset MyPerson = createObject("component",
   "com.mycompany.Person")>
<cfset MyPerson.init("Bob Smith")>
```

The Person.cfc also has a display() method that outputs the person's name directly to the browser. Some developers seem to feel that embedding output in a CFC is a bad practice. I use it sparingly but for some tasks returning the value only to instantly display it in the browser seems unnecessary, especially when the "output" variable on the <CFFUNCTION> tag makes it clear that this is intended functionality.

Association and Aggregation

To understand composition we need to start with two basic premises of

CFDJ Advertiser Index

ADVERTISER	URL	PHONE	PAGE
ACTIVEPDF	WWW.CFCONF.ORG/CFUN-03	866.GoToPDF	4
CFDYNAMICS	WWW.CFDYNAMICS.COM	866.233.9626	25
COOLFUSION	WWW.COOLFUSION.COM	631.737.4668	9
EDGE WEB HOSTING	WWW.EDGEWEBHOSTING.NET	1.866.EDGEWEB	3
FUSETALK	WWW.FUSETALK.COM/NEW	866.477.7542	2
HOSTMYSITE.COM	WWW.HOSTMYSITE.COM	877.215.HOST	27
INTERMEDIA.NET	WWW.INTERMEDIA.NET	800.379.7729	52
IVIS TECHNOLOGIES	WWW.IVIS.COM	602.346.5045	15
LINUX BUSINESS & TECHNOLOGY	WWW.SYS-CON.COM	888.303.5282	35
MACROMEDIA	WWW.MACROMEDIA.COM/GO/CERTIFICATION1/		21
MACROMEDIA	WWW.MACROMEDIA.COM/GO/DEVNET-CFDJ		51
NETQUEST	WWW.NQCONTENT.COM		11
NEW ATLANTA COMMUNICATIONS	WWW.NEWATLANTA.COM		6
PACIFIC ONLINE	WWW.PACIFICONLINE.COM	877.503.9870	39
PAPERTHIN	WWW.PAPERTHIN.COM	800.940.3087	31
SOUTHWEST SUPPORT SOLUTIONS		866.654.5443	19
SYS-CON SUBSCRIPTIONS	WWW.SYS-CON.COM/2001/SUB.CFM	888.303.5282	43
TERATECH	WWW.CFCONF.ORG/CFUN-03/	301.881.1440	17
WEB SERVICES EDGE WEST 2003	WWW.SYS-CON.COM	201.802.3069	23

PACIFIC ONLINE

www.pacificonline.com

"pure" object-oriented programming. The first is that everything is an *object*. The second is that objects are (often) "composed" of other objects. There are two types of object composition: association and aggregation. The distinction is dependent on the relationship between the internal object(s) and its container object:

- ***Composition by association:*** When an object, or in our case CFC, is composed of independent and externally visible child objects. That is, a "Table" object is made up of three or four "Leg" objects and a "TableTop" object. "Leg" objects can also make up a "Chair" object so it's not always directly coupled together with "TableTop". The "Leg"(s) and "TableTop" exist without the need for the Table. In a CFC we would have properties that are themselves an instance of a CFC. This is the basis of an association.
- ***Composition by aggregation:*** Allows a client object to treat both single objects and groups of objects, or composite objects, uniformly and often polymorphically. Usually the parent object instantiates any contained child objects that are encapsulated within the parent CFC and are therefore only accessible through whatever interface the parent CFC makes available. The Composite design pattern primarily relates to aggregation.

If we take a look at Figure 1 we will see a classic composition pattern made up of two kinds of objects: components and composites.

The top-level parent class is AComponent.cfc, which defines some common behaviors and properties. In this case it has a single method called display(). The capital "A" prefix is a naming convention used to indicate that the class is abstract and cannot itself be instantiated. We do this because some CFCs are used as placeholders for the many CFCs that will inherit from them. Also inheriting from AComponent.cfc is SomeComponent.cfc, which is a single concrete (meaning it can be instantiated) object which is not itself a composite. Also inheriting from AComponent.cfc is AComposite.cfc, which defines behaviors and properties for "composite" classes that inherit from it.

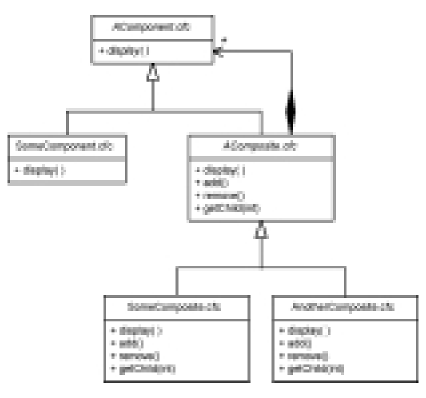


Figure 1: The classic Composite pattern UML diagram

SomeComposite.cfc and AnotherComposite.cfc, which are also concrete composite classes, inherit from AComposite.cfc overriding methods when appropriate. An overriding method is when a child CFC implements a method with the same name as a method in the CFC it inherits from, thus overriding the parent CFC method's code and replacing it with code customized for the current CFC.

Since every class implements a display() method, a CFC or calling page can treat all the classes uniformly when calling display(). For concrete someComponent.cfc it may output something to the browser. For either concrete composite class it may output something itself and/or it may call the display() method for every child object contained within it. The fact that each CFC object itself can vary the actual processing is what makes it polymorphic.

The Employee Table

Our example of using composition and the Composite pattern in ColdFusion begins with a simple Access database. The database itself is quite simple. We have a single table called Employee with at least the following fields: Employee_ID, Name, Title, Salary, HourlyRate, EmployeeType, and Boss_ID. The Boss_ID is the Employee_ID of that person's boss and both are represented by UUIDs. We also could have used autonumbers or another numeric primary key. With this information we know who everyone's boss is. Perhaps the chairman doesn't have a Boss_ID or maybe it's his Employee_ID stating that he is his own boss. Either will work. We are using the former. This is shown in Figure 2.

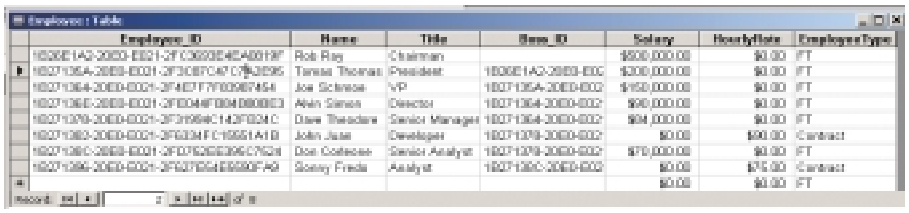
If you want to install the source code for this article you will need to download the Access database at sys-con.com/coldfusion/sourcec.cfm.

Aggregate CFC Hierarchy

Our CFC hierarchy begins with the aforementioned Person.cfc, which defines common behaviors and properties of all people, whether employees or contractors. Extending Person.cfc is AEmployee.cfc, which is an abstract base class for all Employee classes to inherit from. Two CFCs extend AEmployee.cfc. These are FTEmployee.cfc and ContractEmployee.cfc. These are both "concrete" classes, but only FTEmployee.cfc is a composite object (because a contractor can't be a boss). Each defines some behaviors and properties and override methods when appropriate.

Please note: this is a slightly different "model" for implementing the Composite pattern than we saw in Figure 1. There are a number of different implementations and as always the implementation depends on the language, platform, and business requirements. Our CFC hierarchy is shown in Figure 3.

Again, as we saw before, a calling page can treat all the classes uniformly when calling display(). For concrete FTEmployee.cfc it may output something to the browser. For that concrete composite class it may output something itself and/or it may call the display() method for every child object contained within it. Either way the calling page only needs to know that an object inherits from Person.cfc to be able to call a display() method. This again is the nature of polymorphism.


Employees : Table

Employee_ID	Name	Title	Boss_ID	Salary	HourlyRate	EmployeeType
1E0DE1A2-20E0-E021-2FC3020E4EA0819F	Rob Ray	Chairman		$500,000.00	$0.00	FT
1027135A-20E0-E021-2F3C87C47C[illegible]2E95	Teresa Thomas	President	1E0DE1A2-20E0-E0C	$200,000.00	$0.00	FT
10271364-20E0-E021-2F4E7F7F0096[illegible]454	Joe Schmoe	VP	1027135A-20E0-E02	$150,000.00	$0.00	FT
1027136E-20E0-E021-2F0044F0B4D80BE3	Alain Simon	Director	10271364-20E0-E02	$90,000.00	$0.00	FT
10271378-20E0-E021-2F31994C142F024C	Dave Theodore	Senior Manager	10271364-20E0-E02	$84,000.00	$0.00	FT
10271382-20E0-E021-2F6234FC19951A1B	John Jane	Developer	10271378-20E0-E02	$0.00	$90.00	Contract
1027138C-20E0-E021-2FD7626E396C762	Don Corleone	Senior Analyst	10271378-20E0-E02	$70,000.00	$0.00	FT
10271396-20E0-E021-2F627E64E6090FA9	Sonny Freda	Analyst	1027138C-20E0-E02	$0.00	$75.00	Contract
				$0.00	$0.00	FT

Record: 2 of 8

Figure 2: The employee database

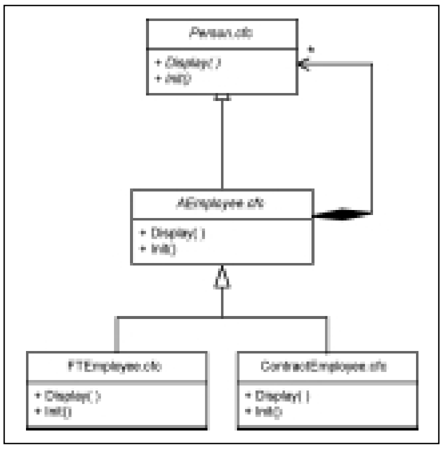


Figure 3: Our modified Composite example

AEmployee.cfc

AEmployee.cfc extends Person.cfc and adds three additional properties. These are Title, Salary, and Boss_ID, which are unique to employees. These directly correlate to the Employee table. Salary defaults to $0.00 and Title and Boss_ID default to empty string (""). The CFC still contains only the two methods from Person.cfc although it overrides them both. In an actual implementation many more methods would be implemented here but are left out for simplicity's sake. Listing 1 shows the code for AEmployee.cfc.

The init() method does a lot more than in Person.cfc. First of all it takes only a single argument, which is a UUID (as we'll demonstrate later). It then retrieves the "Employee" information from our Employee table before setting it into the appropriate fields in the My scope. And the display() method outputs more than just the name as it does in Person.cfc. It also outputs the Title and Salary. These methods are both overridden in ContractEmployee.cfc and FTEmployee.cfc but they are here nonetheless to act as defaults for any additional CFCs that may later extend it.

ContractEmployee.cfc

ContractEmployee.cfc extends AEmployee.cfc and adds one additional Property – HourlyRate – which applies only to contract employees. The CFC still contains only the two methods from Person.cfc and AEmployee.cfc although again, it overrides them both. Listing 2 shows the code for ContractEmployee.cfc.

The only real difference in the init() method is that it sets My.HourlyRate, but not My.Salary, which does not really apply to contract employees. The display() method changes are also minor. It adds the word "(Contractor)" to the first line and displays the hourly rate instead of salary on the second. We'll explain the purpose of the prefix argument later.

FTEmployee.cfc

FTEmployee.cfc also extends AEmployee.cfc and adds two additional properties that are fairly significant. Listing 3 shows the code for FTEmployee.cfc.

The ChildrenSet property is a query recordset that holds information retrieved during the init() method relating to all the people working for the current Employee instance. At this point, you may wonder if this could gather up a large amount of data. It could, such as if we asked for information about the chairman. But then this is really no different than a CFQUERY doing a join. We choose not to do that in this example in order to get the benefits of treating the records like objects. This is an important distinction worth observing.

ChildrenObjectArray is an array that will contain CFC objects, which represent this Employee's subordinates. ChildrenObjectArray is defaulted to an empty array. The actual CFC instances in ChildrenObjectArray may be of FTEmployee.cfc, ContractEmployee.cfc or any type that inherits from AEmployee.cfc. Again, here we begin to see the polymorphic behavior that makes OO design so powerful.

The init() method in FTEmployee.cfc does a lot more than in any of the previous CFCs. Again it takes only a single argument, which is a UUID. It then uses the var keyword of CFSET to declare as local to the init method two variables which we will be using later in the method. It then retrieves the "Employee" information from our Employee table for the passed in Employee_ID which it sets into the appropriate fields in the My scope. It then queries the Employee table again to retrieve all the records for employees whose Boss_ID matches this employee's Employee_ID. These records are then set into the ChildrenSet Property.

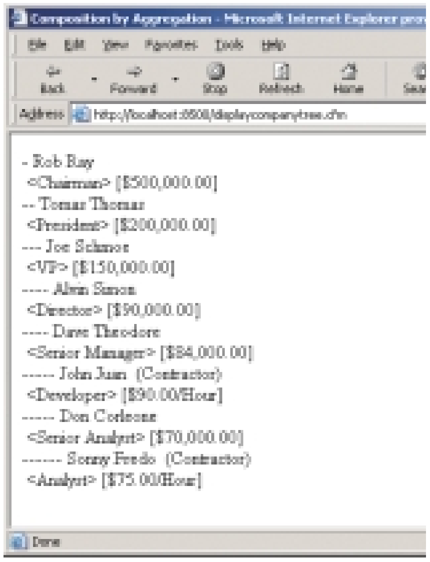
If the ChildrenSet.recordcount is greater than 0, we loop over the ChildrenSet query creating the appropriate CFC object for each employee in the recordset. Note that this is an example of reuse and recursion. Rather than try to handle some predefined number of levels of children and creating code for each level, we will instead call the very init() method that we're already processing, this time to gather up the data for the current employee's subordinates. This could lead to those subordinates calling the method for their subordinates, and so on. These CFC instances are then added to the ChildrenObjectArray property. init.

For all the complexity going on in init() the display() method is relatively simple. It outputs the current employee's name, title, and salary, then increments the argument.prefix which will be passed in to subordinate display() methods for formatting purposes.

Now we see what the prefix argument is used for. It's a value which, as we recurse into lower levels of the hierarchy, will have dashes increase to represent the depth of the level we're at, as shown in Figure 4. Then it loops over the ChildrenObjectArray property calling each CFC object's display() method, never knowing whether they are ContractEmployee.cfc instances or FTEmployee.cfc instances, which is the nature of polymorphism.

Testing the Code

To test the display() method and ensure it is truly polymorphic, we simply instantiate a FT employee CFC object passing in the Employee_ID of the chairman of this smallish company. The code snippet is below and also in Listing 4 (see Figure 4).


Composition by Aggregation - Microsoft Internet Explorer pro
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Sea
Address http://localhost:8500/displaycompanytree.cfm
Done

```
- Rob Ray
 <Chairman> [$500,000.00]
-- Tomas Thomas
 <President> [$200,000.00]
--- Joe Schmoe
 <VP> [$150,000.00]
---- Alvin Simon
 <Director> [$90,000.00]
---- Dave Theodore
 <Senior Manager> [$84,000.00]
----- John Juan  (Contractor)
 <Developer> [$90.00/Hour]
----- Don Corleone
 <Senior Analyst> [$70,000.00]
------ Sonny Fredo  (Contractor)
 <Analyst> [$75.00/Hour]
```

Figure 4: Our "primitive" employee hierarchy display

```
<cfset Employee =
createObject("component","com.mycompany.FTEmpl
oyee").init("1B26E1A2-20E0-E021-
2FC3693E4EA0819F")>
<cfoutput>#Employee.display()#</cfoutput>
```

Once we have created a reference to the FTEmployee CFC object we call the init() method passing in Mr. Rob Ray's Employee_ID. This returns us the object instance. We then call its display() method. As we know the FTEmployee.cfc is a composite CFC and display() will not just output Rob Ray but also information relating to anyone who works for Rob Ray and in fact anyone who works for them and so on. This recursive display() allows us to output a primitive organizational hierarchy. If we were to insert anyone else's Employee_ID we would get their info displayed and all of their subordinates' info ad infinitum.

Conclusion

Design patterns often rely on object composition to determine the best solution for a given situation. Composition by association, the ability to have one CFC object as a property of another, and composition by aggregation, as seen in the Composite pattern, are critically important to our ability to associate common objects and their related patterns. For us as developers, design patterns open us up to new ways of doing things. As I believe Sir Isaac Newton once said, "If I have seen further, it is because I have stood on the shoulders of giants... ." Allowing ourselves to build on the well-established work of others allows us to reach new heights as well.

About the Author

Brendan O'Hara is one of the coauthors of Advanced Macromedia ColdFusion MX Application Development, *published by Macromedia Press. Brendan has a Macromedia ColdFusion MX Developer Certification along with Java and Linux certifications from Penn State University. Brendan has just been named to Team Macromedia for ColdFusion. He is a ColdFusion, Java, and .NET software architect in the Philadelphia suburbs.*

brendantohara@yahoo.com

Listing 1

```
<cfcomponent extends=" Person" displayname="Abstract Employee">
    <cfparam name="My" type="struct" default="#structNew()#">
    <cfset My.Title = "">
    <cfset My.Salary = "$0.00">
    <cfset My.Boss_ID = "">
    <!--- methods --->
    <cffunction name="init" access="public" returntype="string">
            <cfargument name="Employee_ID" required="Yes" default="">
            <cfquery name="qGet" datasource="Employee">
                        Select *
                        From Employee
                        Where trim(Employee_ID) = '#arguments.Employee_ID#'
            </cfquery>
            <cfset My.Name = qGet.Name>
            <cfset My.Title = qGet.Title>
            <cfset My.Salary = qGet.Salary>
            <cfset My.Boss_ID = qGet.Boss_ID>
<cfreturn this>
    </cffunction>
    <cffunction name="display" access="public" returntype="void" output="Yes">
            &nbsp;- #My.FirstName# #My.LastName#<br>
            &nbsp;&lt;#My.Title#&gt;&nbsp;[#dollarformat(My.Salary)#]
    </cffunction>
</cfcomponent>
```

Listing 2

```
<cfcomponent extends="com.mycompany.AEmployee" displayname="ContractEmployee">
  <cfparam name="My" type="struct" default="#structNew()#">
  <cfset My.HourlyRate = "">
  <!--- methods --->
  <cffunction name="init" access="public" returntype="struct">
    <cfargument name="employee_id" required="Yes" default="">
    <cfquery name="qGet" datasource="Employee">
      Select *
      From Employee
      Where trim(employee_id) = '#trim(arguments.employee_id)#'
    </cfquery>
    <cfset My.Name = qGet.Name>
    <cfset My.Title = qGet.Title>
    <cfset My.HourlyRate = qGet.HourlyRate>
    <cfset My.Boss_ID = qGet.Boss_ID>
    <cfreturn this>
  </cffunction>
  <cffunction name="display" access="public" returntype="void" output="Yes">
    <cfargument name="prefix" required="Yes" default="-">
    #arguments.prefix#&nbsp;#My.Name#&nbsp;&nbsp;(Contractor)<br>
    &nbsp;&lt;#My.Title#&gt;&nbsp;[#dollarformat(My.HourlyRate)#/Hour]<br><br>
  </cffunction>
</cfcomponent>
```

Listing 3

```
<cfcomponent extends="com.mycompany.AEmployee" displayname="FullTimeEmployee">
  <cfparam name="My" type="struct" default="#structNew()#">
  <cfset My.ChildrenSet = "">
  <cfset My.ChildrenObjectArray = Arraynew(1)>
  <!--- methods --->
  <cffunction name="init" access="public" returntype="struct">
            <cfargument name="employee_id" required="Yes" default="">
            <cfset var x = 0>
            <cfset var thisEmployee = "">
            <cfquery name="qGet" datasource="Employee">
                        Select *
                        From Employee
                        Where trim(employee_id) = '#arguments.employee_id#'
            </cfquery>
            <cfset My.Name = qGet.Name>
            <cfset My.Title = qGet.Title>
            <cfset My.Salary = qGet.Salary>
            <cfset My.Boss_ID = qGet.Boss_ID>
            <cfquery name="qGetAgain" datasource="Employee">
                        Select *
                        From Employee
                        Where trim(Boss_ID) = '#arguments.employee_id#'
            </cfquery>
            <cfset My.ChildrenSet = qGetAgain>
            <cfif My.ChildrenSet.recordcount GT 0>
            <cfloop query="My.ChildrenSet">
                        <cfset temp = "com.mycompany.#EmployeeType#Employee">
                        <cfset thisEmployee = createObject("component",temp).
                                       init(Employee_ID=My.ChildrenSet.Employee_ID)>
                        <cfset My.ChildrenObjectArray[My.ChildrenSet.currentrow] = thisEmployee>
            </cfloop>
             </cfif>
            <cfreturn this>
    </cffunction>
    <cffunction name="display" access="public" returntype="void" output="Yes">
            <cfargument name="prefix" required="Yes" default="-">
    #arguments.prefix#&nbsp;#My.Name#<br>
            &nbsp;&lt;#My.Title#&gt;&nbsp;[#dollarformat(My.Salary)#]<br>
    <cfset arguments.prefix = "#arguments.prefix#-">
    <cfloop from="1" to="#arraylen(My.ChildrenObjectArray)#" index="theChild">
    #My.ChildrenObjectArray[theChild].display(arguments.prefix)#
    </cfloop><br>
    </cffunction>
</cfcomponent>
```

Download the Code...
Go to www.coldfusionjournal.com

A LIMITED TIME SAVINGS OFFER FROM SYS-CON Media

SUBSCRIBE TODAY TO MULTIPLE MAGAZINES

AND SAVE UP TO $400 AND RECEIVE UP TO 3 FREE CDs!

RECEIVE YOUR DIGITAL EDITION ACCESS CODE INSTANTLY WITH YOUR PAID SUBSCRIPTIONS

TO ORDER •Choose the Multi-Pack you want to order by checking next to it below. •Check the number of years you want to order. •Indicate your location by checking either U.S., Canada/Mexico or International. •Then choose which magazines you want to include with your Multi-Pack order.

Pick a 3-Pack, a 6-Pack or a 9-Pack

☐ 3-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.
☐ 6-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.
☐ 9-Pack	☐ 1YR	☐ 2YR	☐ U.S.	☐ Can/Mex	☐ Intl.

☐ **Linux Business & Technology**

U.S. - Two Years (24) Cover: $143	You Pay: $79.99 /	Save: $63 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $39.99 /	Save: $32
Can/Mex - Two Years (24) $168	You Pay: $119.99 /	Save: $48 + FREE $198 CD
Can/Mex - One Year (12) $84	You Pay: $79.99 /	Save: $4
Int'l - Two Years (24) $216	You Pay: $176 /	Save: $40 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **Java Developer's Journal**

U.S. - Two Years (24) Cover: $144	You Pay: $89 /	Save: $55 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $49.99 /	Save: $22
Can/Mex - Two Years (24) $168	You Pay: $119.99 /	Save: $48 + FREE $198 CD
Can/Mex - One Year (12) $84	You Pay: $79.99 /	Save: $4
Int'l - Two Years (24) $216	You Pay: $176 /	Save: $40 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **Web Services Journal**

U.S.- Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **.NET Developer's Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **XML-Journal**

U.S. - Two Years (24) Cover: $168	You Pay: $99.99 /	Save: $68 + FREE $198 CD
U.S. - One Year (12) Cover: $84	You Pay: $69.99 /	Save: $14
Can/Mex - Two Years (24) $192	You Pay: $129 /	Save: $63 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $89.99 /	Save: $6
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **WebLogic Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

☐ **ColdFusion Developer's Journal**

U.S. - Two Years (24) Cover: $216	You Pay: $129 /	Save: $87 + FREE $198 CD
U.S. - One Year (12) Cover: $108	You Pay: $89.99 /	Save: $18
Can/Mex - Two Years (24) $240	You Pay: $159.99 /	Save: $80 + FREE $198 CD
Can/Mex - One Year (12) $120	You Pay: $99.99 /	Save: $20
Int'l - Two Years (24) $264	You Pay: $189 /	Save: $75 + FREE $198 CD
Int'l - One Year (12) $132	You Pay: $129.99 /	Save: $2

☐ **Wireless Business & Technology**

U.S. - Two Years (24) Cover: $144	You Pay: $89 /	Save: $55 + FREE $198 CD
U.S. - One Year (12) Cover: $72	You Pay: $49.99 /	Save: $22
Can/Mex - Two Years (24) $192	You Pay: $139 /	Save: $53 + FREE $198 CD
Can/Mex - One Year (12) $96	You Pay: $79.99 /	Save: $16
Int'l - Two Years (24) $216	You Pay: $170 /	Save: $46 + FREE $198 CD
Int'l - One Year (12) $108	You Pay: $99.99 /	Save: $8

☐ **WebSphere Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

☐ **PowerBuilder Developer's Journal**

U.S. - Two Years (24) Cover: $360	You Pay: $169.99 /	Save: $190 + FREE $198 CD
U.S. - One Year (12) Cover: $180	You Pay: $149 /	Save: $31
Can/Mex - Two Years (24) $360	You Pay: $179.99 /	Save: $180 + FREE $198 CD
Can/Mex - One Year (12) $180	You Pay: $169 /	Save: $11
Int'l - Two Years (24) $360	You Pay: $189.99 /	Save: $170 + FREE $198 CD
Int'l - One Year (12) $180	You Pay: $179 /	Save: $1

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

3-Pack

Pick any 3 of our magazines and save up to $27500 Pay only $175 for a 1 year subscription plus a FREE CD

- 2 Year – $299.00
- Canada/Mexico – $245.00
- International – $315.00

6-Pack

Pick any 6 of our magazines and save up to $35000 Pay only $395 for a 1 year subscription plus 2 FREE CDs

- 2 Year – $669.00
- Canada/Mexico – $555.00
- International – $710.00

9-Pack

Pick 9 of our magazines and save up to $40000 Pay only $495 for a 1 year subscription plus 3 FREE CDs

- 2 Year – $839.00
- Canada/Mexico – $695.00
- International – $890.00

Subscribe Online Today www.sys-con.com/2001/sub.cfm

SYS-CON MEDIA

Macromedia Announces Flash MX Data Connection Kit

(*San Francisco*) – Macromedia, Inc., has announced the Macromedia Flash MX Data Connection Kit, which provides prebuilt connections to a range of data sources to jumpstart development of data-aware rich Internet applications. The kit, available for $299, includes Macromedia Firefly Components acquired from CyberSage Software, as well as a developer edition of Macromedia Flash Remoting MX.

"Developers have asked us to make it easier to connect to data sources within Macromedia Flash applications," said Norm Meyrowitz, president of products, Macromedia. "This offering is part of an ongoing commitment by Macromedia to provide developers with higher-level components and patterns to enable a simpler, faster way to build rich Internet applications."

The Macromedia Firefly Components provide a framework for accessing, displaying, and updating data within Macromedia Flash applications. For rapid data access, the Firefly component architecture includes specialized connectors and resolvers that integrate with multiple data sources including XML, Microsoft SQL Server, and Macromedia Flash Remoting. For consistent data display, the components include common visual building blocks that separate presentation from application logic. For simplified data updates, shadowing technology used by the components enables efficient saving of information to original data sources.

"Since Macromedia Firefly Components are 'data-aware,' they provide a mechanism for performing database operations without writing any code on the client side. This environment has greatly reduced our development time and saved us several hundred hours of development costs," said Joe Pryzbylkowski, lead member, engineering staff, Lockheed Martin Corporation. "The rapid application development environment that these tools provide has allowed us to design, develop, and deliver versions of our application ahead of cost and schedule. Without a doubt, it was due to Macromedia Flash, Macromedia Flash Remoting MX, and Macromedia Firefly Components."

In addition, the Macromedia Firefly Remoting connector is tightly integrated with ColdFusion and offers developers advanced functionality such as server-side introspection and automatic updates.

The developer edition of Macromedia Flash Remoting MX (included in the data connection kit) can be used with Macromedia Firefly Components to access additional data sources such as Macromedia ColdFusion MX, Microsoft .NET, Java, and SOAP-based Web services. The developer edition of Macromedia Flash Remoting is a fully functional, multi-IP enabled server, allowing teams to test and integrate this powerful functionality.

The Macromedia Flash MX Data Connection Kit is priced at $299 for commercial users and $199 for the education channel. The kit is available for purchase from the Macromedia Online Store (www.macromedia.com/store). For more information on the Macromedia Flash MX Data Connection Kit, visit www.macromedia.com/go/dck.

Flash Keeps Classes Open for More than 60 Hong Kong Schools

(*San Francisco*) – Faced with the Severe Acute Respiratory Syndrome (SARS) virus outbreak, Hong Kong Baptist University turned to technology to keep students learning and able to attend virtual classes after the government shut down schools to contain the virus. With the help of Macromedia Flash Communication Server MX 1.5, classes that are open to public access continued for more than 6,500 students from 60 elementary and secondary schools.

"Macromedia Flash Communication Server MX 1.5 makes it possible to deliver multiway audio, video, and real-time data to so many students at the same time, all streaming through the tiny Macromedia Flash Player on a student's PC," said Dr. Alex C.W. Fung, head of the education studies department and the school administration and management system training and research unit, Hong Kong Baptist University. "Instructors' lesson plans and learning materials are uploaded and stored on the server ready to use later during the real-time Webcasts of each lesson," he said. "Teachers reduced content development time significantly by using Macromedia Flash MX to quickly convert all teachers' existing PowerPoint presentation files before uploading to VITLE," he said.

Hong Kong Baptist University was already test-running the Virtual Integrated Teaching and Learning Environment (VITLE) using Macromedia Flash Communication Server MX and Macromedia ColdFusion MX within its campus, but the availability of Macromedia Flash Communication

CFUN-03

CFUN-03, the fifth annual ColdFusion conference, is set for June 21–22, 2003, in Washington DC. Last year's event was sold out with 300 attendees.

This year's conference has 18 nationally known speakers including Charlie Arehart, Ray Camden, Hal Helms, Michael Smith, Michael Dinowitz, Simon Horwith, Shlomy Gantz, and our own Robert Diamond, the editor-in-chief of ***CFDJ***! There will be four tracks with subjects for beginner ColdFusion, Advanced ColdFusion, MX Integration, and Empowed Programming (Fusebox, Project Management, etc.).

There will also be an exhibit zone where you can learn about the latest MX products. Last year's sponsors included Macromedia, PaperThin, New Atlanta, New Riders, ***ColdFusion Developer's Journal***, Open Demand, Fusion Authority, and FuseTalk.

Joe Hayes, an atttendee at CFUN-02, offered organizers the following feedback: "I wanted to drop you a note and say thanks for the CFUN-02 conference. I have been to many conferences in my 21 years of professional programming, and this by far was the best conference I have ever attended. The information provided by each and every presenter was top notch. I can't wait for CFUN-03 and if preregistration were available today, I would sign up today! This conference is an incredible value and to me was worth many times the price. Take care and thanks again for all the great speakers!"

CFUN-03 is run by MDCFUG and TeraTech (the winner of the 2002 ***CFDJ*** Readers' Choice Award for best consulting company). www.cfconf.org

Server MX 1.5 made it possible to stream presentations and live audio and video to students across the territory.

Macromedia, Microsoft, and other technology companies are providing software, support, and infrastructure to help this "Classes Suspended but Learning Continues" initiative using VITLE. Fung developed this improved version of VITLE using an ASP model with Macromedia Flash Communication Server MX 1.5. It was launched in jus two working days with the help of a Macromedia engineer.

"We're starting to see more and more examples of the Internet being used as a two-way communication medium," said Rob Burgess, chairman and CEO, Macromedia. "Hong Kong Baptist University has created a live virtual classroom where students are able to see the professor on the screen and literally conduct the class through this difficult time. It's a great example of the Internet delivering more than HTML experiences."

To see streaming video examples of VITLE in action, visit www.macromedia.com/go/vitle. VITLE is available at www.iLearn.com.hk.

Cast Your Vote for Your Favorite CF Products/Services

Now's your chance to honor your favorite ColdFusion products/services by voting in the 2003 ColdFusion Developer's Journal Readers' Choice Awards competition now in full swing at www.sys-con.com/coldfusion/readerschoice2003. Widely referred to as "The Oscars of the Software Industry," the Readers' Choice Awards program has become the most respected industry competition of its kind. Voting began on March 1, and will continue until August 30, 2003. Winners wil be announced at Web Services Edge 2003 West, in Santa Clara, CA, September 30–October 2.

FuseTalk Inc. Announces Release of FuseTalk Professional Edition 4.0

(*Ottawa, ON*) – FuseTalk Inc., the leading provider of premium forum and collaboration tools for Macromedia ColdFusion environments, has announced the release of FuseTalk Professional Edition 4.0.

Building on FuseTalk's reputation for rich features, high performance, and value pricing, FuseTalk Professional Edition's newest release contains customer-driven features and enhancements that benefit both administrators and forum users.

"FuseTalk Professional Edition 4.0 brings even greater speed and enhanced performance to our forum," said Anand Lal Shimpi, founder and president of AnandTech, which operates one of the largest forums in the world. "With new usability features like subcategories, private messages, and RSS support, FuseTalk Professional Edition 4.0 gives our forum members an even better collaborative experience."

"We've always taken a customer-oriented approach to deciding what to deliver in new releases," said Dominic Plouffe, FuseTalk Inc.'s vice president of R&D. "That approach has worked well again in this case. With FuseTalk Professional Edition 4.0 we believe that we're delivering a top-notch product to customers who need to be sure that their collaborative environment is fast, highly stable, and exciting to use."

FuseTalk Professional Edition 4.0 is available at www.fusetalk.com.

cf community

—continued from page 7

loop when there was a CFML function that would extract the values from the recordset for him. But no one hit the nail on the head like Douglas when he suggested that the best practice was to make the database perform the majority of the work. Chris reported back that in the end what he ended up doing was the following:

```
<!--- get email list --->
<cfquery name="list" datasource="#Request.App.Db#">
SELECT DISTINCT CONCAT(email,'\r\n') AS email
FROM `mail_subscriptions`
#optlist#
</cfquery>

<cffile action="write" file="#ExpandPath("users.lst")#"
output="#valuelist(list.email,"")#">
```

By making the database do the majority of the work, Chris was able to cut the 180-second execution time down to 14 seconds, which is very reasonable for the nightly routine. Once again, the database shines through.

The lesson here wasn't just that making the database do work for you can save valuable time and other resources (we all know that but sometimes forget), but also that there are often several CFML variations that offer different advantages, drawbacks, and performance differences, when solving the same problem. While many of these options are valid programmatic solutions, the first place a developer should look for answers is often the database.

ColdFusion User Groups

For more information go to…
http://www.macromedia.com/cfusion/usergroups

New England

New Hampshire
Northern N.E. MMUG
www.mmug.info

Massachusetts
Boston, MA CFUG
www.cfugboston.org

Rhode Island
Providence, RI CFUG
www.ricfug.com

Vermont, Montpelier
Vermont CFUG
www.mtbytes.com/dfug/index.htm

Midatlantic

New Jersey, Raritan
Central New Jersey CFUG
www.freecfm.com/c/cjcfug/index.cfm

New York
Albany, NY CFUG
www.anycfug.org

New York
Long Island, NY CFUG
www.licfug.org

New York
New York, NY CFUG
www.nycfug.org

New York
Rochester, NY CFUG
www.roch-cfug.org

New York
Syracuse, NY CFUG
www.cfugcny.org

Pennsylvania, Harrisburg
Central Penn CFUG
www.centralpenncfug.org

Pennsylvania
Philadelphia, PA CFUG
www.pacfug.org

Pennsylvania
Pittsburgh, PA CFUG
www.orbwave.com/pghcfug

Pennsylvania
State College, PA CFUG
www.cfug-sc.org

Southern

Alabama
Birmingham, AL CFUG
www.bcfug.org

Alabama
Huntsville, AL CFUG
www.nacfug.com

Delaware, Dover
Delaware CFUG
www.decfug.org

Delmarva, Dover
Delmarva CFUG
www.delmarva-cfug.org

Florida
Gainesville, FL CFUG
http://plaza.ufl.edu/aktas

Florida
Orlando, FL CFUG
www.cforlando.com

Florida
Tallahassee, FL CFUG
www.tcfug.com

Florida
Tampa, FL CFUG
www.tbcfug.org

South Florida
South Florida CFUG
www.cfug-sfl.org

Georgia, Atlanta
Atlanta, GA CFUG
www.acfug.org

Georgia, Atlanta
Georgia CFUG
www.cfugorama.com

Georgia
Columbus, GA CFUG
www.vcfug.org

Kentucky
Louisville, KY CFUG
www.loulexcfug.com

Louisiana
New Orleans, LA CFUG
www.nocfug.org

Maryland
Annapolis, MD CFUG
www.ancfug.com

Maryland
Baltimore, MD CFUG
www.cfugorama.com

Maryland
Broadneck H. S. CFUG
www.cfug.broadneck.org

Maryland, Bethesda
Maryland CFUG
www.cfug-md.org

Maryland
California, MD CFUG
http://smdcfug.org

North Carolina
Charlotte, NC CFUG
www.charlotte-cfug.org

North Carolina
Fayetteville, NC CFUG
www.schoollink.net/fcfug/

North Carolina
Raleigh, NC CFUG
www.ccfug.org

Oklahoma
Oklahoma City, OK CFUG
http://idgweb4.ouhsc.edu/cfug

Oklahoma
Tulsa, OK MMUG
www.tulsacfug.org

Tennessee
Memphis, TN CFUG
http://cfug.dotlogix.com

Tennessee
Nashville, TN CFUG
www.ncfug.org

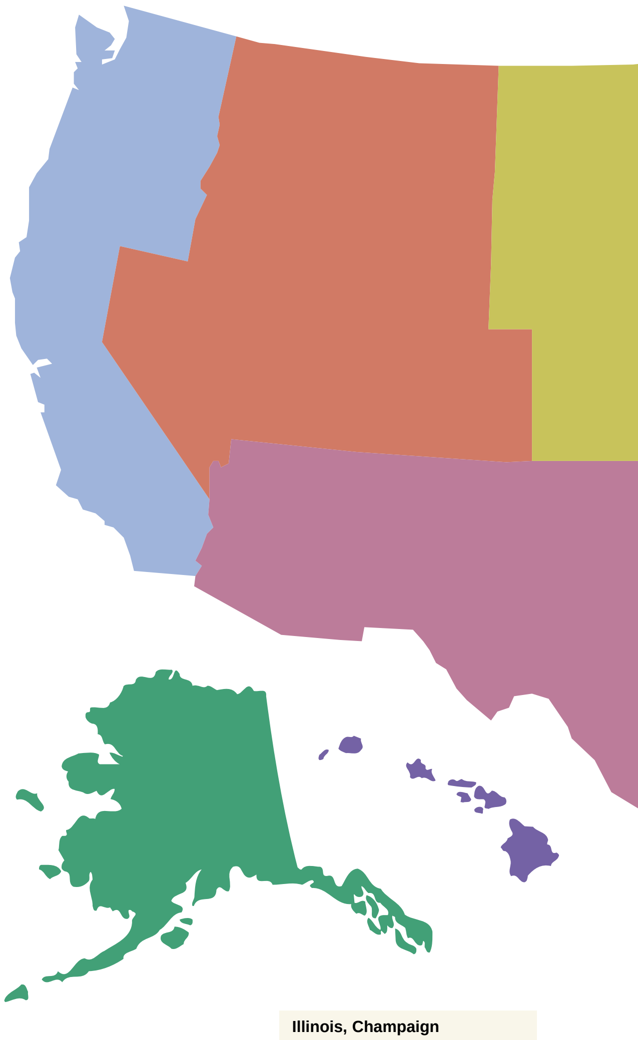
Virginia
Hampton Roads CFUG
www.hrcfug.org

Virginia
Northern Virginia CFUG
www.cfugorama.com

Virginia
Richmond, VA CFUG
http://richmond-cfug.btgi.net

Virginia, Roanoke
Blue Ridge MMUG
www.brmug.com

Washington D.C.
Washington, D.C. CFUG
www.cfugorama.com



Midwest

Northern Colorado
Northern Colorado CFUG
www.nccfug.com

Colorado Hamilton M.S. CFUG
http://hamilton.dpsk12.org/teachers/team-c/intro.html

Illinois, Champaign
East Central Illinois CFUG
www.ecicfug.org

Illinois, Chicago
Chicago, IL CFUG
www.cfugorama.com

Indiana
Indianapolis, IN CFUG
www.hoosierfusion.com

Indiana, South Bend
Northern Indiana CFUG
www.ninmug.org

Iowa
Des Moines, IA CFUG
www.hungrycow.com/cfug/

Michigan, Dearborn Mmaniacs CFUG http://ciwebstudio.com/mmania/mmania.htm

Michigan, East-Lansing
Mid Michigan CFUG
www.coldfusion.org/pages/index.cfm

Minnesota, Minneapolis
Twin Cities CFUG
www.colderfusion.com

Rockies

Montana, Helena
Montana CFUG
http://montanacfug.site-o-matic.com

Nevada
Las Vegas, NV CFUG
www.sncfug.com

Utah
Salt Lake City, UT CFUG
www.slcfug.org

Wyoming, Jackson
Wyoming MMUG
www.wycfug.org

West Coast

California
Bay Area CFUG
www.bacfug.net

California, Fresno
Central California CFUG
www.cccfug.com

California, Inland Empire,
Inland Empire CFUG
www.sccfug.org

California
Los Angeles CFUG
www.sccfug.org

California
Orange County CFUG
www.sccfug.org

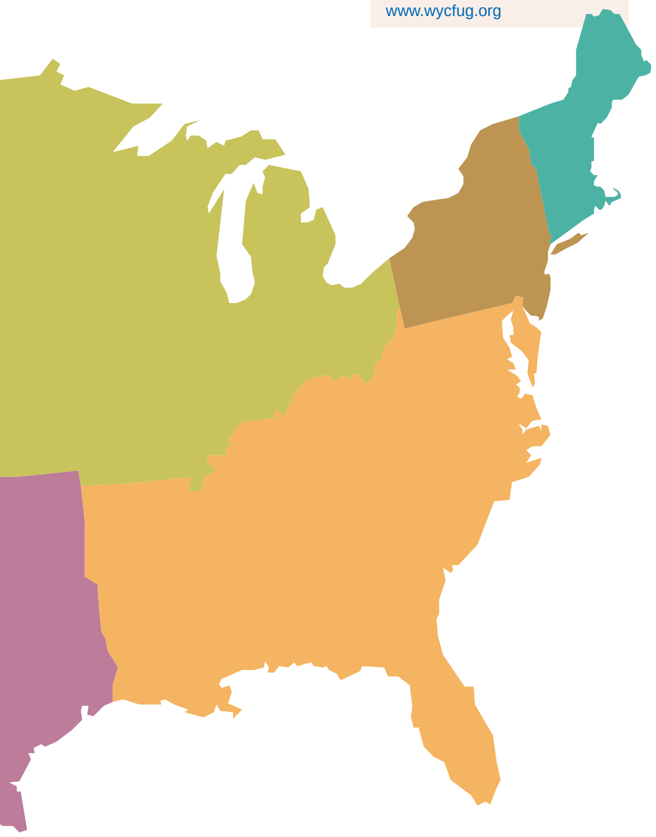
California
Sacramento, CA CFUG
www.saccfug.org

California
San Diego, CA CFUG
www.sdcfug.org

Southern California
Southern California CFUG
www.sccfug.org

Oregon
Eugene, OR CFUG
www.EugeneCFUG.org

Oregon
Portland, OR CFUG
www.pdxcfug.org



Alaska

Alaska
Anchorage, AK CFUG
www.akcfug.org

Hawaii

Hawaii, Honolulu
Hawaii CFUG
http://cfhawaii.org

Missouri
Kansas City, MO CFUG
www.kcfusion.org

Missouri
St. Louis, MO CFUG
www.psiwebstudio.com/cfug

Nebraska
Omaha, NE CFUG
www.necfug.com

Ohio, Columbus
Ohio Area CFUG
www.oacfug.org

Ohio
Mid Ohio Valley MMUG
www.movcfug.org

Wisconsin
Milwaukee, WI MMUG
www.metromilwaukee.com/usr/cfug

Southwest

Arizona
Phoenix, AZ CFUG
www.azcfug.com

Arizona
Tucson, AZ CFUG
www.tucsoncfug.org

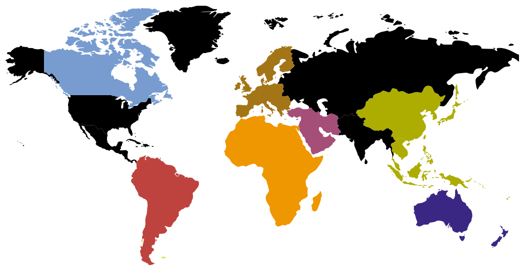
Texas
Austin, TX CFUG
http://cftexas.net

Texas
Dallas, TX CFUG
www.dfwcfug.org

Texas
San Antonio, TX CFUG
http://samcfug.org

About CFUG's

ColdFusion User Groups provide a forum of support and technology to Web professionals of all levels and professions. Whether you're a designer, seasoned developer, or just starting out – ColdFusion User Groups strengthen community, increase networking, unveil the latest technology innovations, and reveal the techniques that turn novices into experts, and experts into gurus.



Africa

South Africa, Cape Town
Cape Town, South Africa CFUG
www.coldfusion.org.za

South Africa, Johannesburg
Johannesburg, South Africa CFUG
www.coldfusion.org.za

Asia

Malaysia, kuala lumpur
Malaysia CFUG
www.coldfusioneer.com

Thailand
Bangkok, Thailand CFUG
http://thaicfug.tei.or.th

Japan, Urayasu-city Japan CFUG
http://cfusion.itfrontier.co.jp/jcfug/jcfug.cfm

Australia

Australia-Melbourne
Australian CFUGs
www.cfug.org.au

Canada

Canada
Edmonton, AB CFUG
http://edmonton.cfug.ca

Canada
Kingston, ON CFUG
www.kcfug.org

Canada
Montreal, QC CFUG
www.kopanas.com/cfugmontreal

Canada
Ottawa, ON CFUG
www.cfugottawa.com

Canada, Ottawa (HS group)
Ecole Secondary CFUG
www.escgarneau.com/gug

Canada
Toronto, ON CFUG
www.cfugtoronto.org

Canada
Vancouver, BC CFUG
www.cfug-vancouver.com

Canada
Vancouver Island CFUG
www.cfug-vancouverisland.com

Europe

Belgium, Brussels
Belgium CFUG
www.cfug.be

Central Europe, Munich
Central Europe CFUG
www.cfug.de

Finland, Helsinki
Finland CFUG
www.cfug-fi.org

France, Valbonne
France CFUG
www.cfug.fr.st

Germany, Frankfurt
Frankfurt CFUG
www.cfug-frankfurt.de

Ireland
Belfast, Ireland CFUG
www.cfug.ie

Ireland
Cork, Ireland CFUG
http://viewmylist.com/cork

Italy, Bologna
Italy CFUG
www.ingenium-mmug.org

Sweden
Gothenburg, Sweden CFUG
www.cfug-se.org

Switzerland
Martin Bürlimann, Switzerland CFUG
www.cfug.ch

United Kingdom, London
UK CFUG
www.ukcfug.org

United Kingdom
Northern England CFUG
www.cfug.org.uk

Middle East

Pakistan Edu, Lahore Cantt
Pakistan Educational CFUG
www.cfeugpakistan.org

Pakistan, Lahore
Pakistan CFUG
www.cfugpakistan.org

Saudi Arabia, Riyadh
CFUG Saudia
www.cfugsaudia.org

Turkey, Izmer
Turkey CFUG
www.cftr.net

South America

Brazil
Rio de Janerio CFUG
www.cfugrio.com.br

Making the Case for CFML

We've been on the right track all along

Many BlueDragon customers tell us they're being asked to defend their choice of CFML (ColdFusion Markup Lanuage) over JavaServer Pages (JSP). They need help making the argument that CFML and J2EE work well together, and that perhaps CFML is a better choice for presentation layer technology than JSP for developing J2EE Web applications. They know instinctively it's the right choice, but aren't quite sure how to build the case.

By Vince Bonfanti

As more organizations are standardizing on J2EE (and .NET) the issue of defending CFML will only become more urgent. The good news is that you've made the right choice with CFML. In this edition of **BluePrints**, we'll talk about why CFML can be a better choice than JSP for J2EE Web applications (in a future column, we'll take a look at why CFML can be the right choice of presentation layer technology for ASP.NET applications).

We'll see some of the reasons to stick with CFML, and identify some of the frequent arguments held against traditional CFML applications. We'll show how CFML has changed to address some of those arguments, as well as how design choices play a part in developing effective applications.

Then we'll look at the other side of the coin, discussing some of the common challenges of working with JSP – challenges that simply aren't an issue for CFML developers. We'll also see how JSP itself is changing, with the introduction of the Expression Language (EL) in the upcoming JSP 2.0 specification, and with the recently released JSP Standard Tag Library (JSTL). Indeed, the case can be made that JSP is looking more and more like CFML. Clearly we've been making the right choice all along.

This column will conclude with a brief discussion of why it doesn't even have to be a choice about "CFML versus JSP" at all. With both BlueDragon and CFMX, not only can your CFML apps coexist with your JSP/servlet apps, but the two platforms can benefit from tight integration with each other (BlueDragon also provides tight integration with the Microsoft .NET Framework, to be discussed in a future article).

CFML? What's CFML?

If you're a JSP developer who has happened upon this article, it may be useful to put CFML in context. Sometimes, people have gotten a misguided notion of what CFML is. Let's take a moment to recap. CFML really solves the same problems as JSP: it's a server-side scripting language for dynamic generation of Web content (typically HTML).

Created originally for ColdFusion Server (by Allaire, later acquired by Macromedia), CFML is no longer a single-vendor, proprietary language. New Atlanta's BlueDragon product line offers several implementations of CFML, which allow you to run CFML on both J2EE and .NET servers, adding the possibilities of tight integration with native applications.

Anyone who's worked with CFML will agree that it's an incredibly productive language for Web application development. Designed from its inception to make dynamic database-driven applications easy for HTML coders, CFML has always been easy to understand. CFML code looks a lot like HTML. More important, CFML packs a lot of power into a few lines of code. Here's an example of a typical query and output process:

```
<CFQUERY NAME="Products" DATASOURCE="ProdDB">
      SELECT * FROM Products
</CFQUERY>

<H1> Product List</H1>

<CFOUTPUT QUERY="Products">
#ProductName# $#Price# <BR>
</CFOUTPUT>
```

Of course, there are all the usual programming constructs (setting and using variables, control flow with ifs and looping, user defined functions, and more.) CFML is more than "just a language." It's also a framework with support for application and session variables, flexible database access and connection pooling, query caching, and more.

At the same time it's also a "high level" language incorporating features that on other platforms might require incorporating additional libraries or other products. Ben Forta's December 2002 ***CFDJ*** article, "But It's Free" (www.sys-con.com/coldfusion/article.cfm?id=541) discusses this aspect of CFML's richness in the context of comparing it with open source and other "free" platforms. As someone once said, CFML makes easy things easy and difficult things possible.

CFML's ease of use benefits not only new programmers but also coders responsible for maintaining someone else's application. Whether or not you're a CFML programmer, you can often readily tell what a CFML template is doing. It's almost self-documenting.

What's Not to Like?

While the tag-based nature makes it easier for non-programmers to pick up, some purists may argue that it's an ineffective mechanism for programming. Many developers don't find that it hampers their abilities at all. Still, the language has evolved to include an available CFSCRIPT tag to allow coding in a scripting approach based on ECMAScript (which is itself derived from JavaScript, though in the case of CFSCRIPT it's entirely server-side processing).

CFML's ease of use can be a double-edged sword though, as applications become more complex. As the earlier example shows, a query and its embedded SQL usually appear on the top of a page processing the query results. If the query page is the result of a form submission (such as a search page), it's typical to also throw the input validation onto the top of that same page. Some even go so far as to put the form and its processing on the same page, testing at runtime to know whether to show or process the form.

These methods are familiar to CFML developers, and they reflect the very growth of applications that start out simple and soon encompass more and more functionality. CFML makes it easy to prototype an approach, and there are times when its simplicity also makes it ideal for getting applications out the door quickly.

Still, the mix of business logic and display on a single page can have its drawbacks in larger and more sophisticated applications. And many don't even think about the problem because it's just always "how things were done" in CFML.

Indeed, to carry this scenario further, it's easy for any developer (especially a junior one) to create an application (especially a complex one) that doesn't perform well or won't scale. That's less often an indictment of the language than one of the developer's skills. (All those who've seen a JSP page run amuck doing too much and mixing presentation and logic, raise your hands.

Even so, many will also argue that there are times when segregating your applications into presentation and logic layers is overkill. Purist arguments make for lively e-mail list debates and academic discussions, but the reality for some organizations and for some applications is that the job needs to be done as quickly as possible. In such cases, CFML is just so much easier to get started with.

One final argument in favor of CFML, for a shop that already has a large base of CFML code, is that there's a lot to be said for the knowledge gained and the investment made in those applications. Migrating away from such apps will often be quite an expensive effort. We'll even see later that rather than convert the apps, you may want to explore integrating CFML with JSP applications.

JSP: The Challenges and the Changes

If you or your organization are set to proceed with JSP development anyway, or if you want some justification for holding off on it, let's take a look at some of JSP's challenges, as well as some recent changes in JSP development that influence this debate.

The first thing to consider, as a current CFML developer, is that getting into JSP isn't going to be a trivial process. Sure, JSP itself is a rather limited language and you don't really have to understand Java in order to edit/create JSP templates, but certainly you won't be able to do too much on a JSP page without knowing Java.

The primary goal of the JSP approach was to make it easy for HTML coders to edit presentation pages, with the thinking that the back-end processing (query processing, data validation, etc.) would be handled by a Java developer in a servlet or bean.

Unfortunately, many JSP developers are as unfamiliar as CFML developers with the notion of segregating page processing into layers. They too start to do more in the JSP page than just "presentation." The downside is that if they begin to embed Java code (the scripting language for JSP) onto the page, it will no longer be as easy to understand, edit, and debug.

Not only will a non-Java HTML coder have difficulty with the page, but even a novice Java developer and possibly an experienced developer will find it challenging.

This raises another point against JSP: its error reporting and handling leave a lot to be desired. Because a JSP template is first parsed and then compiled, then finally run, there are lots of moving parts that could make it difficult to track down the source of a problem. When you start to add more Java code to a template, it only exacerbates the problem.

Even if JSP developers start to follow the best practices recommendations of segregating an application into layers, they then have decisions such as how to encapsulate the "model," and even then whether to use beans or EJBs. Then there's the question of whether and how to use servlets in addition to JSPs, such as for controllers, filters, etc. Add in the wide range of J2EE APIs available, and a "JSP" application can quickly grow into a very complex beast.

Most CFML developers are used to developing an application completely by themselves. With so many moving parts, JSP apps often grow to involve many different developers with responsibility for each part. So more than just having to learn Java, they have to become familiar with all the possible layers and how to share these responsibilities. It takes a lot more effort to understand all the implications and choices available.

For a discussion of some of these challenges, see the article, "Is Complexity Hurting Java?" by Jason Weiss in ***CFDJ***'s sister magazine, ***Java Developer's Journal***. It's in the July 2002 issue at www.sys-con.com/java/article.cfm?id=1658.

Another take on the debate in that magazine, from the July 2001 issue, is "JSP: You Make the Decision," by Jon Stevens, at www.sys-con.com/java/article.cfm?id=1080. Part of this article is focused on introducing Turbine, one of many templating languages (including WebMacro, FreeMarker, and more) that have been proposed over the years as alternatives to JSP for Java Web application developers.

In supporting the value of these alternatives over traditional JSP, Stevens points out many of the challenges with JSP, reiterating the problem of embedding Java code within templates. He laments that there's no way to prevent that occurrence in the JSP spec.

Stevens also points out that taglibs (JSP custom tag libraries) are an interesting solution, in that they provide a way to segregate such scripting into custom tags that are perhaps easier for non-Java developers to understand. But he laments that because developers are free to come up with their own, they may be reinventing the wheel. He recognizes efforts to create standardized taglibs, but at the time he wrote the article (nearly two years ago), such efforts were in their infancy.

JSP 2.0: Looking More Like CFML

A lot has changed since Stevens wrote that article. Indeed, the relatively new JSP 2.0 specification really changes the landscape of JSP programming quite a bit. We don't have room here to explain things but suffice it to say that the changes are pretty significant. The new expression language (EL) makes it so that referring to variables in JSP becomes a lot more like referring to variables in ColdFusion (though they're surrounded by ${..} rather than # signs).

It also leans far more toward using less scripting, which addresses the concerns raised previously. It even offers a means to prevent inclusion of scripting in the page. Clearly, they're listening to the folks above complaining. You can learn more about JSP 2.0 at http://java.sun.com/products/jsp.

The JSP Standard Taglib (JSTL) introduces a new set of tags for setting and using variables, control flow with ifs and looping, and performing queries with embedded SQL. Hey! That sure sounds a lot like CFML!

Indeed, an argument could be made that JSP 2.0 is an admission of failure of the original JSP spec and even that it's struggling to become more like CFML. Again, we've been on the right track all along.

Even with all these "improvements," there's still going to be more to creating a typical JSP/servlet application than just getting into JSP 2.0.

After hearing all this, are you sure you want to move to JSP after all? Is it really clear what it will buy you? If it's becoming more like CFML, is it worth all the pain and expense of converting your applications to JSP? Do you really need the complexity of layers, integration of moving parts, and involvement of many developers that is typical of JSP applications?

Not an Either/Or Decision

When faced with the incursion by other technologies like JSP and .NET it's easy to fall into a trap of thinking that "defending" CFML is about choosing (or forestalling) a wholesale move to the other platform. It's not an either/or decision.

Whether yours is a JSP 1.x or 2.0 shop, or you're considering an alternative templating language, what if you could keep your current CFML applications and then consider whether to convert any aspects to JSP and/or the rest of the J2EE suite of APIs?

With both BlueDragon Server JX and CFMX Enterprise Server, you can run JSPs and servlets alongside your CFML templates and can even natively integrate the two. You can share session and application variables between CFML templates and JSP/servlets, transfer control and "include" information among them, and more.

And if you need to deploy your CFML applications on a J2EE server (such as WebSphere, WebLogic, Sun ONE, or others), there are J2EE editions of both BlueDragon and CFMX. BlueDragon's deployment approach offers advantages that will be important to most J2EE shops.

I've discussed all these points in my previous columns, – "It's Not ColdFusion – It's J2EE" and "CFML Forever," – both available at www.newatlanta.com/products/bluedragon/editorials.cfm.

CFML As a New J2EE Presentation Layer

Of course there will be those in the J2EE community who will argue that all the improvements in JSP 2.0 are just whitewash over a fundamentally challenged foundation. I agree.

What's needed is a mature, tag-based language, one that's equally suited to layered development or to simpler designs that may embed the model and the view at once. And one that doesn't become more difficult to debug and maintain when you do that. Hmm. Where might they find one? Indeed, one can absolutely make the argument that CFML can serve as a better presentation layer for J2EE applications.

Conclusion

If you have existing CFML assets, you can continue to leverage them effectively because CFML is a valid alternative as a native J2EE Web scripting language. You have many options.

You can stick with your CFML running on our free BlueDragon Server or CFMX Pro and simply ignore JSP/servlets. You can purchase BlueDragon Server JX ($549) or CFMX Enterprise ($4999), keeping your CFML applications, and also run JSP/servlets alongside them, deciding at your own pace when to migrate or simply integrate those applications. Or if you want to deploy your CFML applications on a J2EE server, you can do that as well with BlueDragon/J2EE (starting at $2,499) or CFMX for J2EE (starting at $3,399). With these you can get all the benefits of such a platform while still preserving your investment in CFML.

In my next column, I'll introduce you to BlueDragon for .NET Servers. It's the only solution that provides all these same benefits of native integration in a .NET server environment. With our BlueDragon/J2EE and BlueDragon.NET products, we offer the only way to implement your CFML applications on both the major application server platforms.

Indeed, we've made the first Web application development language in the entire industry that's compatible across such a wide range of server platforms. That's the way to protect your investment in CFML.

About the Author

Vince Bonfanti is president and co-founder of New Atlanta Communications, developers of Java- and CFML-based server products. A charter member of Sun's Java Servlet API and JavaServer Pages Expert Groups, Vince has been a JavaOne speaker and a contributor to Java trade magazines and online publications. He has also been a featured speaker at Toronto's CFNorth and Washington's CFUN conferences as well as at local ColdFusion User Groups around the country.

vince@newatlanta.com

MACROMEDIA

www.macromedia.com/go/devnet-cfdj

INTERMEDIA.NET

www.intermedia.net