U.S. $8.99 (Canada $9.99)

ColdFusion Developer's Journal

ColdFusionJournal.com

October 2002 Volume: 4 Issue: 10

NEW ATLANTA

WWW.NEWATLANTA.COM

ColdFusion Developer's Journal

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

What a Difference a Year Makes!

BY **ROBERT DIAMOND**

As October comes around, it's time for another Macromedia DevCon, generally regarded as the largest gathering of ColdFusion developers each year, along with users of Macromedia's other technologies. The theme of this year's conference, which reflects the strategy we've watched unfold throughout this past year, is clearly one of integration. It's no secret that Macromedia would like to be the provider of all your Web application design and development needs, and many of this year's classes cover just that. For those not wanting to put all their eggs in one basket, however, they remain dedicated to working with existing technologies, as we can see in this month's news. I am a great supporter of integration, and hope these efforts work as well as planned.

This has certainly been a banner year for ColdFusion, with the release of ColdFusion MX. Rewritten from the ground up over a period of more than two years, and with only a handful of visible quirks thus far, it's the largest and one of the more extended complete releases in recent memory.

Year 2002 has been a banner one for Macromedia, and they've raised the bar to make 2003 even better. At DevCon we should gain our first glimpse of what's ahead. I for one hope to see increased stability, along with the usual commitment to new feature and integration potential.

Macromedia has also released three new versions of the ColdFusion MX Application Server, which can now run on top of some of the most popular Java/J2EE servers. The new versions are specifically optimized for IBM's WebSphere Application Server, the Sun ONE Application Server, and Macromedia JRun. (The distinction should be made here that this is the full release of JRun as opposed to that shipping with the regular version of CFMX). The new releases will enable ColdFusion developers to work even more closely with the Java language. One of the greatest benefits that customers should realize is the ability to deploy higher-load applications, further entwining the scalability, reliability, and power of the Java language with ColdFusion.

Another side effect we should see is the increased use of ColdFusion on enterprise Java platforms, bringing new developers over to ColdFusion. The ability to run it on top of their existing server setups should be a great boon and increase the global use of CF. Look for reviews of all three new versions in upcoming issues of *ColdFusion Developer's Journal*.

• • •

This issue is filled with all kinds of useful tips to make your development lives easier and more productive. **Dennis Baldwin** writes on getting connected with Flash debugging, using the NetConnection Debugger to simplify your Flash/CF application development. **Jason Clark**, of FuseTalk, Inc., gives some more great tips on building application login systems, packaging your apps for sale, and more. **Steve Drucker** takes a detailed look at CommonSpot's Content Server 3.0 from PaperThin, with a glowing review for those looking for content management tools.

Charlie Arehart serves up Part 1 of a series on how to precompile CFMX templates, **Simon Horwith** provides some useful SQL info in "Tales from the List," and **Steve Bryant** covers error handling in JavaScript for those using it in conjunction with CF code.

Last, but never least, **Hal Helms** explores a "young-minded" approach to building applications.

Enjoy, and stay tuned to *CFDJ* and www. ColdFusionJournal.com for total coverage of this year's Macromedia DevCon.

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

ROBERT@SYS-CON.COM

By Dennis Baldwin

Get Connected with Flash Debugging

With Flash Remoting on the rise we're beginning to see a plethora of advanced Flash applications hit the Web. I'm not talking about silly Web site intros or loading screens, but full-blown Web applications such as shopping carts, e-mail clients, content management systems, and expense reporting. Flash Remoting allows developers to do so much more with applications by utilizing Flash MX as the UI while serving up dynamic content from ColdFusion MX (or any application server that supports Flash Remoting).

As Flash applications become increasingly more complex, the harder they become to maintain and debug. If it weren't for debugging, applications would produce all sorts of errors and become very unstable. Through extensive testing and debugging, applications can run as expected and generate fewer headaches for both the developer and end user.

NetConnection Debugger: An Overview

We're going to take a look into an incredible tool that Macromedia provides for Flash developers known as the NetConnection Debugger (NCD), shown in Figure 1. The NCD simplifies application development by providing an intuitive UI that displays diagnostic information of the communication process between Flash, Flash Remoting, and CF. Developers can monitor data that is sent to and from the server such as query results (recordsets), arrays, structures, strings, integers, and so on. To access the NCD, the Flash Remoting components need to be installed as well as Flash MX and ColdFusion MX. For more information on Flash Remoting and downloading the components be sure to check out www.macromedia.com/software/flash/flashremoting/. To access the NCD go to Window > NetConnection Debugger within the Flash IDE.

As pretty as the NCD may seem, it serves no purpose without the necessary code to activate it and put it to work. To view the communication process, the NetDebug.as ActionScript file needs to be included in the Flash movie. This file is installed when the Flash Remoting components are installed and can be included using the following code:

```
#include "NetDebug.as"
```

Make sure to include this file in Frame 1 of the main timeline in your movie. This provides the code to activate the NCD and the methods to trace results to the NCD window. You don't need to worry about the location of this include file if you've successfully installed the Flash Remoting components. Flash initially looks in the local directory for the include file and then in the <Flash MX Install Dir>\Configuration\Include directory. The latter directory serves as a global location for includes when compiling Flash movies.

Note: It's good practice to include this file only when debugging your applications. When releasing your application to a production environment, be sure to comment this line out or remove it completely as this include file adds about 10KB to your compiled Flash file size.

If you're at all familiar with previous versions of Flash, you'll remember there was no simple way to debug applications. The problem could exist on either the client or server, and a series of steps had to be taken to help pinpoint the error. With the NCD all activity between the client and server can be monitored. Therefore any errors generated will be displayed in the NCD, which will keep the debugging process centralized and make it that much easier.

Configuring the Debug Application

Now that I've ranted and raved about how great the NCD is, let's look at a few working examples. First we'll retrieve data from CF via Flash Remoting and view the results in the NCD. You'll need to download the source files from www.flashcfm.com/cfdj/debug.zip and unzip them on your development machine. I recommend creating a directory labeled "cfdj" directly under your wwwroot. If you intend to install the files elsewhere, change the reference to the service object in debug.fla. The ActionScript code is located in Frame 1 of the AS layer. For the directory structure mentioned above, your code would look like:

```
myServiceObj = gw.getService("cfdj.debug", this);
```

The directory cfdj is directly under the Web root and debug points to the ColdFusion component (CFC). We establish a service connection with this component and can then make calls to different methods within the component. If you'd placed the install files in a directory like \wwwroot\flashremoting\cfdj\, then your service connection would look like:

```
myServiceObj = gw.getService("flashremoting.cfdj.debug",
this);
```

You should also confirm that the URL variable is set correctly. For most local development configurations this will point to localhost but your configuration may vary. Once these settings are confirmed, you're ready to test the application and see how the NCD works.

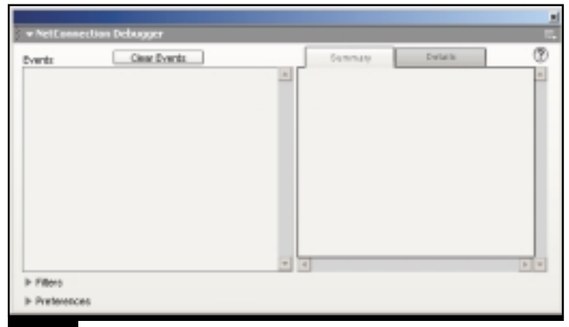

FIGURE 1 NetConnection Debugger

NetConnection Debugger at Work

Make sure the NCD panel is activated and then test the movie by going to Control > Test Movie or hitting CTRL-Enter. The first event you'll see listed in the NCD is the connection string made to the Flash Gateway. You should see several buttons in the Flash movie, and we'll start off by retrieving data from CF. If you click the button labeled "retrieve query from server", you'll see a few events displayed in the debugger. These events include a call to the CFC method labeled "getQuery", the time to execute the query, and the results of the query. You'll also notice a trace event, which is a call to the NetDebug.trace method. This method allows you to specify a custom ActionScript object that you want to display in the NCD.

Figure 2 shows the query results sent from CF to Flash via Flash Remoting.

All debugging events are shown in the left-hand panel; summary and details are shown on the right. Icons displayed in front of the events denote whether the event corresponds with the client (Flash) or the server (CF). There's also an icon for general information messages and errors. All property objects are listed in blue, property names in red, and property values in black. The NCD allows you to view all the recordset data passed from CF in a single window. The same applies for other data types, such as arrays and structures. Click the other buttons to see what kind of info is displayed in the NCD.

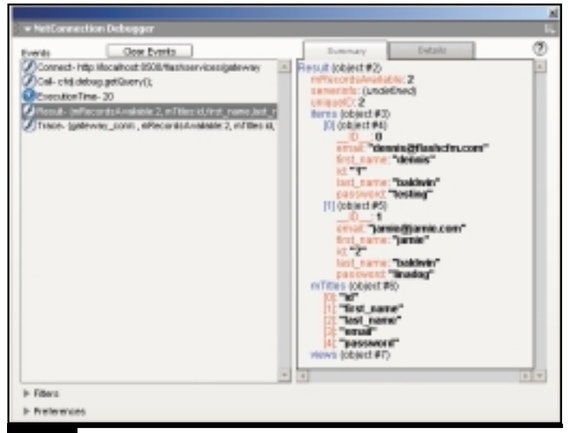

FIGURE 2 Query results sent from CF to Flash via Flash Remoting

Once the service method is called, the information is sent back to a result method in Flash. Since we're calling the getQuery method of the CFC, getQuery_Result will receive the response from the server and we can handle the data accordingly.

In this example we use the NetDebug.trace method to display the query results in the NCD. This can be useful when trying to view all of the query data at once. Listing 1 shows the service objects and their result methods, which trace the data to the NCD.

Now we'll take a brief look at sending data in the other direction, from Flash to CF. The ActionScript code varies a little since we have to construct the data objects on the client side this time. We'll call an imaginary service method named "receive" just for the purpose of generating an error and seeing the results within the NCD. If you click on the button labeled "send recordset to server", you'll see the method call in the events window and then an error should display (see Figure 3). If you look at the summary of the error, there will be a property called description, which generally contains the diagnostic information for the error. In this case the service doesn't contain a method named "receive" and the NCD tells us so.

So instead of having to test for errors directly on the server, Flash Remoting allows the error messages to be pulled into the NCD. This saves time and frustration when debugging applications. Listing 2 shows the necessary ActionScript code to handle passing different data objects from Flash to CF.

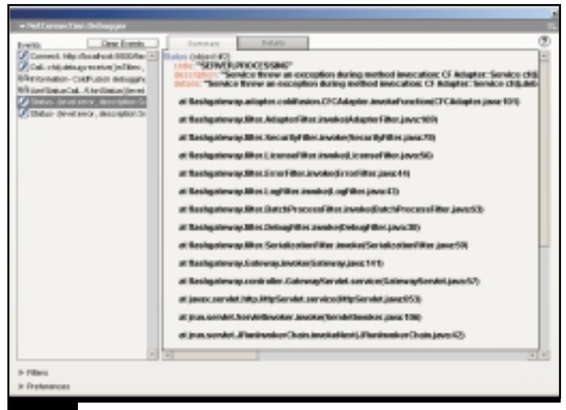

FIGURE 3 NetConnection Debugger displaying an error message

Conclusion

Now you should feel a bit more comfortable with using the NetConnection Debugger and troubleshooting your applications. Try to make a habit of having this panel active when testing Flash Remoting applications; you won't regret it. I can't tell you how much time this nifty little panel has saved me during the debugging process. It helps to keep everything centralized so more of your time will be spent within the Flash IDE and less time will be spent locating errors.

This is just a basic overview of the NCD. Feel free to look into some of the more advanced features, such as Filters and Preferences. Happy debugging!

About the Author

Dennis Baldwin is the lead developer for Eternal Media, a Web/multimedia firm that provides technology solutions for ministries and nonprofit organizations (www.eternal-media.com). He also maintains a couple of online resources for Flash and ColdFusion developers at www.flashcfm.com and www.devmx.com.

@ DENNIS@FLASHCFM.COM

Listing 1: ActionScript code that handles the data passed from CF to Flash

```
// retrieve data from CF based on which button clip is
clicked
function getData(component) {
 if(component == retrieve_query_mc) {
  myServiceObj.getQuery();
 } else if (component == retrieve_array_mc) {
  myServiceObj.getArray();
 } else if (component == retrieve_structure_mc) {
  myServiceObj.getStruct();
 }
}

// query result method
getQuery_Result = function (result) {
 Netdebug.trace(result);
 trace("query received");
}

// array result method
getArray_Result = function (result) {
 Netdebug.trace(result);
 trace("array received");
}

// structure result method
getStruct_Result = function (result) {
 Netdebug.trace(result);
 trace("structure received");
}
```

Listing 2: ActionScript code that handles the data passed from Flash to CF

```
// send data to CF based on which button clip is clicked
function sendData(component) {
 if(component == send_recordset_mc) {
  sendRecordSet();
 } else if (component == send_array_mc) {
  sendArray();
 } else if (component == send_structure_mc) {
  sendStruct();
 }
}

// method called to send recordset to cf
function sendRecordSet() {
 var rs = new RecordSet();
 var temp = {first_name: "dennis", last_name: "baldwin",
email: "dennis@flashcfm.com", password: "testing"};
 rs.addItem(temp);
 temp = {first_name: "jamie", last_name: "baldwin", email:
"jamie@jamie.com", password: "linadog"};
 rs.addItem(temp);
 myServiceObj.receive(rs);
}

// method called to send array to cf
function sendArray() {
 var a = new Array();
 a[0] = "dennis";
 a[1] = "jamie";
 a[2] = "steve";
 a[3] = "juan";
 myServiceObj.receive(a);
}

// method called to send structure to cf
function sendStruct() {
 var o = new Object();
 o.first_name = "todd";
 o.last_name = "rafferty";
 o.email = "todd@devmx.com";
 o.password = "testing";
 myServiceObj.receive(o);
}
```

CODE LISTING

The code listing for this article is also located at

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

Maybe We Should Try a Separation

It can be a good thing

BY **BEN FORTA**

I have been writing and talking about ColdFusion components since before ColdFusion MX shipped. After I explained them in detail in two recent columns (***CFDJ***, Vol. 4, issues 6, 7), quite a few of you asked for practical examples of when and where they should be used. So, once again, let's take a look at CFCs, but this time from a very different angle.

Spot the Problem

To get things going, take a look at the following code (a working example, assuming you installed the ColdFusion sample applications and databases). It reads a user list from a database table and then displays the user names in an HTML unordered list:

```
<!--- Get users --->
<CFQUERY NAME="users"
         DATASOURCE="exam-
pleapps">
SELECT EmployeeID AS UserID,
       FirstName, LastName
FROM tblEmployees
ORDER BY LastName, FirstName
</CFQUERY>

<H1>Users</H1>

<!--- List users --->
<UL>
<CFOUTPUT QUERY="users">
   <LI>#LastName#,
#FirstName#</LI>
</CFOUTPUT>
</UL>
```

So what's wrong with it?

Don't see it? Look again. Carefully.

Still nothing?

Well, maybe there's nothing actually wrong with the code. It will run as is and will work (generating the appropriate output). But while it's syntactically correct, there actually is something very wrong with it.

<CFQUERY> Proliferation

Let's take a step back. Think about the application you last wrote, or the one you're working on right now.

Question: How many <CF QUERY> tags does your code contain? You can guess if you like (although I'm sure your guess will be lower than the actual total).

Next question: How many of the SELECT statements in your various <CFQUERY> tags are similar (or even the same)? You'll likely find that most of your queries interact with the same set of tables. Maybe one retrieves fewer or more columns than another, or sorts results differently, or just passes different WHERE clauses, but beyond that, how many are totally different?

Now for the most important question: What would happen to your code if a database column were renamed? Even more exciting, what would happen if a database schema were updated (as they should be periodically) so that your tables were split into multiple relational tables? Or – and I've saved the best for last – what if the database were changed altogether, maybe switching from one DBMS to another, or moving to an LDAP directory or an XML data store? What would that do to your code?

Think it doesn't happen? It does, and when it does, rewriting client code can be painful. And often the changes aren't made, even though they should be, out of fear of the work involved. So improvement and progress are stymied by code-phobia (I made that word up as the English language didn't seem to contain one as eloquent. Feel free to use it yourself).

The problem is that the very nature of the Web and Web pages, combined with the immediacy of the platform, encourages this type of development (and, I should add, not just in ColdFusion). Chances are, you've written pages that include everything from database queries, HTML output, form validation, all sorts of conditional processing, and business rules, to logic and…you get the idea.

And that's what's wrong with the code I showed you previously:

- *Is that code reusable?* Absolutely not (no, copy-and-paste doesn't equal reusable).
- *Would any UI created in that page be reusable with other data?* Nope, not without lots of tweaking.
- *Would a developer or designer be able to create a better interface without knowing about the underlying data?* Not at all.
- *Would everything break if the database changed?* Most definitely.

For that matter…

- *Could a designer create a richer user experience without even knowing what a database was, or how to write a SQL statement, or how to JOIN tables?* Very unlikely.

Now, once again, think about the app you just finished or the one you're working on. What if there weren't a single <CFQUERY> embedded in the pages that generated output and performed user interaction, would that make your development life easier?

Separation of Presentation and Content

Which bring us to the separation of presentation and content. This isn't a radical new idea (even if the same basic concepts keep resurfacing under new names). The basic concept is that your application needs to be broken into layers (often called *tiers*). Exactly how many layers is up for grabs (there are lots of opinions on this one, and for the most part the discussion is academic), but at a minimum your application should be broken into three distinct tiers:

1. *The back end* (starting with databases)
2. *The front end* (the user interface, be it Web pages, or client-side Flash, or anything else)
3. And in the middle, *the application itself* (often called the *business tier*)

The truth is, the front end should never be tied to any particular database. Each should be free to be improved on (and changed and adapted) without fear of things breaking.

Similarly, you'd never want your business logic in the same page that generates HTML output. Why? Well, as soon as you need an alternate output format (whether it be Flash, XML, a Web service, or even a simple spreadsheet), you'll need to re-create all that logic (or patch your original file).

So, back to our example: the code both retrieved data from a table and formatted it for output. What if I needed an alternate output format? I'd have two choices. Either I'd create a second file (copying the query code) or I'd put a big <CFIF> statement around the output so it could generate both output formats.

While that may be acceptable (and even manageable) for simple code like this, imagine if the query were extremely complex and required all sorts of postprocessing. Then neither option would be really workable. What I'd really want to do is move the database-specific code out of the file that generates presentation into its own file. Then I could have two different presentation files (one for each output format) and have them use the same query by accessing the query file.

In tiered development, content and its presentation are explicitly separated, making each more manageable, more usable, and more reusable.

ColdFusion and Tiered Code

Everything discussed thus far applies to any development in any language, not just ColdFusion. But now let's focus specifically on ColdFusion.

It's been possible to write code like this in ColdFusion for a long time. Between <CFINCLUDE> and custom tags, writing tiered code has been doable. But neither option is ideal. <CFINCLUDE> is ill suited for highly dynamic content (for starters, there's no way to pass variables to an included page or to return results), and custom tags are too cumbersome (no built-in parameter validation, single entry point, can't be introspected and thus can't be shared simply, can't return results, and more).

Which brings us to ColdFusion components. CFCs are designed for just this purpose, building tiered applications. Imagine that you could take all your user-related queries and store them in a single file, then all your product queries and store them in another file, and so on.

Then, to obtain a list of users you could do something like this:

```
<!--- Get user list --->
<CFINVOKE COMPONENT="user"
          METHOD="List"
          RETURNVARIABLE=" users">
```

This code calls a method (a function) named *List* in a component named *user* and returns a result named *users*. What is in the user component? It doesn't matter. Whatever it is will be executed and a result will be returned.

Another way to invoke the component would be:

```
<!--- Get user --->
<CFOBJECT COMPONENT="user"
          NAME="usrObj">
…
<CFOUTPUT>
<!-- Display email address --->
#usrObj.GetEMail(URL.userid)#
</CFOUTPUT>
```

SYS-CON MEDIA

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

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

business development
vp, business development
Grisha Davida grisha@sys-con.com
COO/CFO
Mark Harabedian mark@sys-con.com

advertising
senior vp, sales & marketing
Carmen Gonzalez carmen@sys-con.com
vp, sales & marketing
Miles Silverman miles@sys-con.com
advertising director
Robyn Forma robyn@sys-con.com
advertising account manager
Megan Ring-Mussa megan@sys-con.com
associate sales managers
Carrie Gebert carrieg@sys-con.com
Kristin Kuhnle kristin@sys-con.com
Alisa Catalano alisa@sys-con.com
Leah Hittman leah@sys-con.com

sys-con events
vp, events
Cathy Walters cathyw@sys-con.com
conference manager
Michael Lynch mike@sys-con.com
sales executives, exhibits
Michael Pesick michael@sys-con.com
Richard Anderson richarda@sys-con.com

customer relations
manager, customer relations
Anthony D. Spitzer tony@sys-con.com
customer service representative
Margie Downs margie@sys-con.com

jdj store
manager, customer relations
Rachel McGouran rachel@sys-con.com

web services
web designers
Stephen Kilmurray stephen@sys-con.com
Christopher Croce chris@sys-con.com
Catalin Stancesco catalin@sys-con.com
online editor
Lin Goetz lin@sys-con.com

accounting
assistant controller
Judith Calnan judith@sys-con.com
accounts receivable
Kerri Von Achen kerri@sys-con.com
accounts payable
Joan LaRose joan@sys-con.com
accounting clerk
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

Here the user component is loaded as an object, and a specific method in that object is then invoked – in this example, passing a user ID and obtaining the e-mail address for the specified user.

In both examples the component is now that middle tier – your presentation code talks to the component, and the component talks to the database.

All that's needed then is the component itself.

CFCs as the Middle Tier

As explained in the earlier columns mentioned above, writing CFCs is very easy. (And if you use Dreamweaver MX there's a wizard you can use that does most of the work for you.)

Look at this code:

```
<!--- User component --->
<CFCOMPONENT>

   <!--- List users method --->
   <CFFUNCTION NAME="List"

RETURNTYPE="query">

      <!--- Get users --->
      <CFQUERY NAME="users"

DATASOURCE="exampleapps">
      SELECT EmployeeID AS
UserID,
             FirstName,
LastName
      FROM tblEmployees
      ORDER BY LastName,
FirstName
      </CFQUERY>

      <CFRETURN users>

   </CFFUNCTION>

</CFCOMPONENT>
```

This is a complete (albeit rather simple) component. It defines a single method named *List* (the method used in the prior code snippets). List takes no parameters and returns a query (the same query as used above).

The preceding code would be saved as user.cfc and references to a user component would access this CFC file.

Here's the GetEMail method:

```
<!--- Get e-mail address method
--->
<CFFUNCTION NAME="GetEMail"

RETURNTYPE="string">
   <CFARGUMENT NAME="UserID"
               TYPE="string"

REQUIRED="true">

   <!--- Get user record --->
   <CFQUERY NAME="user"
            DATASOURCE="exam-
pleapps">
   SELECT Email
   FROM tblEmployees
   WHERE EmployeeID
         ='#ARGUMENTS.UserID#'
   </CFQUERY>

   <CFRETURN user.Email>

</CFFUNCTION>
```

This code would be placed into the same user.cfc file (all methods go between the <CFCOMPONENT> and </CFCOMPONENT> tags). It takes a user ID as an argument and returns the e-mail address for the specified user.

This is just the start of it. With all database queries in one isolated location, you can start getting creative. You could:

- Cache data if needed so as to improve performance.
- Maybe even conditionally update cached data based on system load and performance.
- Handle all data conversions and reformatting internally so that UI code need not even be aware of it.
- The list goes on and on.

And it gets better. Dreamweaver MX is CFC aware and can automatically display a tree control listing all CFCs on your ColdFusion server. This allows users to right-click on any CFC or method to see more information about it, and even drag a method into their code to autogenerate the required invocation code. All automatically, and all without having to register anything in Dreamweaver itself.

Note: There's another benefit here, Flash MX integration. But more on that next month.

What Next?

If your application contained not a single <CFQUERY>, your code would be more manageable, more reusable, and even more scalable. I only showed you components encapsulating SELECT statements here, but you'd want to create components that did it all.

Databases and database access is a good starting point. After all, it's what we do mostly with ColdFusion and CFML. But it's just a starting point. Any integration with back-end systems should be tiered, starting with databases but also including:

- COM or CORBA integration
- Server-side HTTP and FTP calls
- Talking to Java APIs, beans, and objects
- XML manipulation
- Web services consumption
- Interacting with ERP systems
- Security and access control
- Business logic and transaction processing
- Any business intelligence (shopping carts, for example)
- …and anything else that isn't tied specifically to presentation

The basic rule is this: if it isn't tied to client code, then it doesn't belong in client-generation code. Simple as that.

Summary

Most of us write spaghetti code, and that has to change. Structured, organized, tiered code is a must. I wouldn't get too hung up on exactly how many tiers are needed and whether they conform to any specific model as long as it gets done. For starters, commit to no more embedded database calls in your output pages – put them in components. Then apply this methodology to existing code. And start thinking about applying this type of development to nondatabase code as well. ColdFusion components make this very easy (and even reward the effort with better performance), so there's no longer any excuse for not doing it. It turns out that separation can indeed be a good thing.

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

By Michael Dinowitz

What Does an E-mail Address Actually Say?

Here's the situation. You're relaxing, reading the latest issue of *CFDJ,* when your boss, significant other, or the voice in the back of your head asks you to write an e-mail retrieval program. This could be for an e-mail feedback system, an error alert manager, or even to handle the huge volume of e-mail that you receive from mailing lists. When this happens, you're going to smile and think that you can just whip out the <CFPOP> tag and use it to retrieve the e-mail you want. That's when the rude awakening hits you. The information you receive from this tag is not in the nice, clean format you're used to in Outlook or Eudora.

Now What?

What you're looking at is the header information of the e-mail message. When we look at an e-mail, we're used to seeing the poster's name, e-mail address, the message itself, and a few other pieces of information. The header of the e-mail contains all this and more. The problem is that it's all in a format that's more easily readable by the computer than by people. Additionally, the format of the information sometimes changes in ways we don't expect.

The obvious answer is to parse out the information. Find some way to define the structure of the information and write code to make it into something clean. This article is designed to do just that. We'll be examining one crucial piece of information stored in the header. This information happens to contain the e-mail address of the sender and sometimes his/her name. We want that information and we want it separated. To do this we'll use a technology called Regular Expressions (RegEx) that allows us to define a pattern and then look for that pattern inside a string. We pay particular attention to the patterns used in e-mail addresses and how to find them in this article.

Let's begin by using a small code sample from a spam catcher. This is an e-mail account that someone would use when posting to unknown locations. It's designed to catch spammers and limit the amount of spam you get to your "real" e-mail account. In order to use such a setup, it's important to look through the mail that it contains every now and again. The code fragment in Listing 1 allows us to read all of the mail messages in such an account and display them. For the purpose of this article, we're only going to get the headers of the messages and then show only the FROM addresses (see Figure 1).

FROM
morthelp012389085933@yahoo.com
<Meridiano_97@MexicoSub.com>
<reacpdm@netscape.net>
Jessica Kincannon <enk2fg7@lycos.com>
"Green Card" <green@abdyesilkart2.com>
bogamil623656946@yahoo.com
ÀÎÑÁÍÁÉÁÍ¼Ê(ÁÓ) <mail@itelx.com>
"Leandro" <Leandro@djhotmail.com>
eden_avellaneda <eden_avellaneda@pal.com.ph>
"SILAGRA TEAM" <contactnow68@hotmail.com>
AkdenizGozMerkezi---0212_6357474@yahoo.com
card25<webmaster@card25.co.kr>

FIGURE 1 FROM addresses

Parsing e-mail addresses in ColdFusion

As we can see from the results, the actual FROM address can be very different. Actually, there are only five basic formats for addresses in the mail header.

```
Name <address@domain.com>
"Name" <address@domain.com>
address@domain.com (Name)
address@domain.com
<address@domain.com>
```

In the first three examples, a plain-text name is sent along with the e-mail address. In the last two, only an e-mail address is sent. This gives us a challenge: how to parse the full e-mail address to get the plain-text name and the actual e-mail address. The answer is actually easier than you'd think. The user-defined function (UDF) in Listing 2 will do this.

Rather Tight, Don't You Think?

Besides being tight, it's also totally incomprehensible to average programmers if they don't know Regular Expressions, UDFs, CFSCRIPT, or the syntax that goes along with it. Let's go through it line by line so we can understand what's going on and why:

1 <CFSCRIPT>: In ColdFusion 5 a UDF has to be written inside a CFSCRIPT block. In CFMX the new CFFUNCTION tag can be used as well. As many people are still using CF5, let's use the "old" method of writing a UDF. CFSCRIPT has to have an opening and closing tag, with all of the actual code written between them. The closing CFSCRIPT tag ends on line 35.

2 Function call: A UDF is defined by using the key word function, followed by the name of the UDF and then open and closed parentheses. Inside the parentheses are the arguments we want to pass to the function. Following the standard rule of using descriptive names, we call the function "ParseEmail" and the argument that it will accept "email". Note that when using this UDF on a page, you can't define a variable called ParseEmail. That name is taken by the UDF.

3 After declaring a UDF, you have to place all of the code for it inside curly braces. This is simply the open brace. Note that I place the brace inline with the function call. This is so it'll be easier to see where something starts. All code inside the braces will be tabbed in by one. Proper tabbing helps greatly in reading and debugging code. The closing tab of this section is on line 34.

5–7 These lines set up variables that will exist only inside the UDF. We do this so we can work with data and not worry about overwriting outside variables. A UDF should be specific in what it accepts and uses. For lines 5–7 all we're doing is creating the variables with a NULL value (i.e., a blank string). Line 9 is the first line where we do some actual work, and this one has to be heavily explained.

9 We're using a standard CF function called REFind(). The RE stands for Regular Expression (RegEx) and the Find means we're trying to find something using the RegEx. A RegEx is a pattern that will be applied to a string or variable. If this pattern is found within the string or variable, then something will be returned to indicate success.

The REFind() function takes from two to four arguments. The first argument is the RegEx itself, the pattern we're looking for. The second is the string (or variable) that the RegEx pattern will be applied to. If only these two arguments are used, then the function will return a number that represents the starting location of the match. If there is no match, a zero will be returned. The third argument is the character location in the string where we want to start. This usually isn't needed unless the fourth argument is used. The fourth argument says that rather than return the numeric location of the start of the match, the function should return a structure that contains the position of the match and its length. It also returns the position and length of any subexpressions (we'll deal with them soon). This fourth argu-

If you haven't used RegEx before, it may look like your editor threw up on the screen, but it actually means something"

ment is a Boolean value.

Note: In the code example we'll be using 1 to represent true. In examples in other places you may see "True" or "Yes". They all mean the same thing.

Looking at RegEx

Now let's examine the RegEx that we're looking for. If you haven't used RegEx before, it may look like your editor threw up on the screen, but it actually means something.

```
^"?([^"<]*)"? *<([^@]+@[^>]+)>
```

I'm not going to try and teach everything about RegEx here. Instead I'm going to describe what each piece does and why. By reading this step-by-step examination, you should learn the basics.

^ Start at the beginning of the string. If the pattern doesn't start at the beginning, it fails. For example, if there is a space before the e-mail address, then it isn't valid.

"? There may or may not be a double quote as the first character in the string. The question mark (?) says that the previous character is optional.

(Start a subexpression, a grouping together of a part of the RegEx so that it's "seen" as a single unit. When the REFind() function has the fourth argument set to true, this subexpression is returned as a separate piece of data in the result structure.

[Start looking for a set of characters. Rather than look for one character to match, this will look for one of many characters.

^ Note that this carat is inside a set declaration (the square brackets). When placed here, it means that we're negating the character set, that is, match any character *other than* whatever follows.

"< These are the characters we're looking for (or not looking for, as the case may be). We want every character that is *not* a double quote or an open bracket.

] Close the character set. The character set is now complete and will match any character that is not a double quote (") or an open angle bracket (<).

***** This is a modifier much like the question mark (?) above. It will take the last character, set, or group and change how many times it can exist in the match. Specifically, it says that the character set that we've defined should exist zero or more times, that is, it will match any number of characters that aren't a double quote (") or an open angle bracket (<).

) This closes the subexpression group. Our complete subexpression says to match any character that isn't a double quote or an open bracket and to match zero or more of those characters. It will continue to match until we run into one of the two characters we don't want.

"? Again, we're looking for a double quote that may or may not exist.

***** Note that before that asterisk is a space. This says that zero or more spaces can exist at this point.

< In three of the e-mail address formats, the actual address is inside brackets.

(Again, we're doing a subexpression group.

[^@]+ As with the set above, we're looking for any character that isn't an at sign (@). The plus means that one or more characters have to exist, that is, there must be a character before the @ sign. Note that we are *not* validating the e-mail address. This function was written to parse e-mail that has already been sent and the address should already be valid. A future article will go over all of the pieces needed to validate an e-mail address.

@ After finding the character(s) before the @, we need to specify the @ sign. All e-mail addresses have one.

[^>]+ Our last set is any character that isn't a closing bracket. One or more have to exist and this will be the domain portion of the address.

) When we're done, we close the subexpression. The entire subexpression says match one or more characters that are not an @ sign, followed by an @ sign followed by one or more characters that isn't a closing bracket.

> After the address there is a closing bracket.

This entire expression will match the following addresses

```
Name <address@domain.com>
"Name" <address@domain.com>
<address@domain.com>
```

In the first case, there are no double quotes, so the '"?' takes care of that. There's a space before the bracket, so the '*' takes care of that. The second case uses the double quotes and the code takes that into account with the '"?' and the negative set containing a double quote '[^"<]'. The third case has no name at all, so the use of the ? (may or may not exist) and the * (may exist zero or more times) came in handy.

Two to Go

Three cases down, but what about the other two? Well, if one of the other address formats existed, the RegEx would fail against it, that is, there would be no match. So how do we detect this?

When a REFind() function is used and return subexpressions is set to true (the fourth argument), then the result of the function is a structure. The structure has two keys in it, Pos and Len. Pos is the start position of the match. Len is how long the match was. Each of the structures contains an array. Each array is equal in length to the other key in the structure, that is, if the Pos array has three items, then the Len array will have three as well. The first item is *always* the entire match. If there was no match then the Pos[1] will be 0 and the Len[1] will be 0. If there were a match, then their values would be the start of the match and its entire length. This is nice, but not what we want. We want to get the position and length of each of the subexpressions. In the example above, if the RegEx matched, then the [2] position of the arrays would contain the first subexpression results (the name) and the [3] position of the array would contain the second subexpression results (the address). Of course, the first subexpression results may be blank, but we'll deal with that later.

Note: No matter what, a REFind() function set up like this will return a structure.

Now that we've defined the RegEx used in line 9, let's move forward.

Back to the Lines

11 We want to know if the subexpressions were returned properly or if we have to try something else. We do this by checking the value of the Pos[1]. If it's zero, then there was no match at all. If it's a positive number, then there was a match and the subexpressions have to exist.

14–15 Using the mid() function, we get the subexpression data and place it in the local variables. The mid() function takes three arguments: a string to work with, a position inside the string to start at, and a length of characters from that position to return. This function is perfect for getting subexpressions.

The first value we'll be getting from the RegEx will be the name portion. As we said earlier, line 7 sets the name variable as being local to the UDF so we don't have to worry about overwriting any other variable called name. The same can be said for the e-mail variable.

The first subexpression was written in such a way as to allow a blank record, that is, there's only an e-mail address but no name. Even if it's blank, there will be a record in the arrays to say that it was at least tried.

18 If the first RegEx ran didn't return any value, then the e-mail address has to be either the third or fourth type listed at the beginning of this article. Both start with the e-mail address without any brackets around it. One also has the poster's name in parentheses after the e-mail address. To parse this out, we use the same steps as the previous RegEx but with two small alterations.

- We'll use the REFindNoCase() version of the tag.
- We'll use a different RegEx pattern.

20 REFindNoCase() Above we used the REFind() function when dealing with the RegEx pattern because we weren't looking to match any characters. When we have to match characters, we have a small issue to deal with: the case of the character. In RegEx the case is important and an uppercase version of a character is different from a lowercase version. To help deal with this, a second version of the function exists: REFindNoCase(). This works the same as the REFind() function, but allows matching of characters regardless of case. The NoCase version is slightly slower (.01 ms or so), as it has to look at both the lower- and uppercase versions of a letter, but we're not really concerned with the minuscule savings.

```
^([-a-z0-9_.]+@[^[:space:]]+)
*\(?([^)]+)?\)?
```

^ Once again we start this match at the beginning of the string [-a-z0-9_.]+. This set is loaded with a lot of interesting things. The first character is a dash (-). Normally in a set this is used to signify a range of characters. The character to the left of the dash is the start of the range and the character on the right is the end. To signify the dash as a character rather than a range indicator, we have to place it as the first character in the set. We next have a range of letters from a to z. Because this RegEx is inside the NoCase version of the function, it actually means the same as a-zA-Z, that is, all characters from lowercase a to lowercase z inclusive, as well as all characters from uppercase A to uppercase Z inclusive. A bit shorter to write than using each character individually.

Note: There are special character sets that could be used in place of the a-z, 0-9, or both together. I've left them out and used the pattern as is to make it easier to learn and understand.

Included in this set are the underbar (_) and the period (.). Usually in RegEx the period is a wild card character, that is, it matches any single character. This is true everywhere *but* within a set. Within a set most

special characters such as the period aren't considered special and are instead seen as their literal character. This means that our set will match a character from a-z, A-Z, 0-9, the dash (-), the underbar (_), and the period (.). Because of the plus (+) after the set, one of these characters *must* exist and may exist as many times as possible. That is, until it comes to a character not in the set.

@ One such character not in the set is the at sign (@). In any e-mail address pattern, one of these must exist. If it doesn't, then there's a big problem with the e-mail address and it may be forged or something else.

[^[:space:]]+ Previously I mentioned special character sets. This is one such set, the space set. When used, it represents any character that can separate text but can't be seen – spaces, tabs, new lines, vertical tabs, form feeds, and carriage returns. Usually these special sets are shown as [[:space:]] to say that it's a set and it's special. We're doing something a little special here by saying that we want any character that *is not* one of those within the special set, so we add the carat (^) inside the first set of brackets. This does the same thing we've done before by negating the set. It just looks strange the way it's presented. Many programmers don't realize this can be done.

^([-a-z0-9_.]+@[^[:space:]]+) This entire match is placed within parentheses so that it will be returned as a subexpression. This will be the e-mail portion of the string. It's possible that there's a name portion as well, so we add some additional code to test for that.

***** There is a space before the asterisk, which means a space can come after our subexpression and more than one can exist. If there is no name portion in this match, then the asterisk allows us not to have a space. If there is one, there will be a space between the e-mail portion and the name. We could use a question mark (?) rather than an asterisk to say that a space may or may not be needed there, but there may be mail programs that use more than one space, so it never hurts to look to the future.

Another Confusing Pattern

We're about to start another one of those really confusing-looking RegEx patterns. Let's take this one really slow:

\(? The backslash (\) before the open parenthesis says that we're looking for a literal parenthesis. The parenthesis is normally the start of a subexpression, but because we're escaping it using the backslash, it's treated like the actual character open parenthesis. The question mark after it says that it's optional. This means that an open parenthesis may or may not exist at this point.

([^)]+)? We're setting up another subexpression here. Because there's a question mark at the end of it, we're saying that the entire expression may or may not exist in our match. The expression itself is a simple set, where we're looking for one or more of any character that isn't a closed parenthesis.

\)? Finally, we're going to match with the closed parenthesis that may or may not exist.

(?([^)]+)?\)? The entire section will match the end of our third e-mail example, a parenthesis that has some text within it followed by a parenthesis. Both parentheses are optional, as is the character string within them. Ugly to look at, but effective for what we want.

21 Again we test the return from the RegEx function to see if it has a value within the arrays. If it fails at this point, the data passed to this UDF did not contain an e-mail address of one of the proper formats.

23–26 At this point we have a successful match and will grab the subexpressions using the mid() function as described above. The use of mid() to get the e-mail is rather straightforward, but the use of it to get the name is a little trickier. In ColdFusion 5, because of the way we structured the last part of the RegEx pattern, the second subexpression (which is the third item in the array) is totally optional. In CFMX it will always exist even though it's set to optional with the question mark modifier (?). Therefore, we have to see how long one of the arrays is and what it contains before using mid() on that position.

25 Here we're just testing if the array len() of the len portion of the array is 3 *and* if it has a value in it other than zero. If so, then the pattern matched a name as well as an e-mail address, and that name should be set to the name variable.

30–31 It's possible at this point that the name variable is still blank even though a proper e-mail address was passed to this UDF. If so, we're going to set the name variable to a value of a single space. This will be important later when we want to display the data.

33 At this point we have a name and an address variable. If no properly formatted e-mail address was passed to this UDF, then both variables are loaded with their default values, which are NULL (a blank string). Otherwise, one or both of the variables have actual text in them. The Return key word within a UDF will take whatever variable or value that's after it and pass it back to where the UDF was called. In this case we're going to cheat. A UDF should generally return only a single piece of data. Since the name and the e-mail address are so tightly interwoven, we're going to return both. To do so, we'll set them up as a list with a comma separating the two pieces of data. The first value will be the name; the second, the e-mail address.

Let's say that this UDF was called to get the e-mail only. To do that, all you need to do is place it within a ListLast() function to get the last value in the list, which will always be the e-mail address. If you want the name, you can use the ListFirst() function. When there's no name portion in the e-mail, a space will be returned because in ColdFusion an empty list entry is ignored.

Now let's run our original code but with this UDF added. To make the example cleaner, we're going to save the above UDF in a separate file and use a CFINCLUDE to add it to the code in Listing 3.

FROM	Name	Address
northelp0123089083933@yahoo.com		northelp0123089083933@yahoo.com
<Meridiano_97@MexicoSub.com>		Meridiano_97@MexicoSub.com
<snacpdm@netscape.net>		snacpdm@netscape.net
Jessica Kincannon <eik2fg7@lycos.com>	Jessica Kincannon	eik2fg7@lycos.com
"Green Card" <green@sbdyeslist2.com>	Green Card	green@sbdyeslist2.com
bogamil6236569946@yahoo.com		bogamil6236569946@yahoo.com
AI'ªAIAEAI¼Ë(AÓ) <mail@telz.com>	AI'ªAIAEAI¼Ë(AÓ)	mail@telz.com
"Leandro" <Leandro@djhotmail.com>	Leandro	Leandro@djhotmail.com
eden_avellaneda <eden_avellaneda@pal.com.ph>	eden_avellaneda	eden_avellaneda@pal.com.ph
"SILAGRA TEAM" <contactnow88@hotmail.com>	SILAGRA TEAM	contactnow88@hotmail.com
AkdenizOrdMerkezi--0212_6357474@yahoo.com		AkdenizOrdMerkezi--0212_6357474@yahoo.com
card25<webmaster@card25.co.kr>	card25	webmaster@card25.co.kr

FIGURE 2 Original e-mail addresses from header

The FROM column in Figure 2 contains the original e-mail addresses from the header as returned by the CFPOP tag. The Name column contains the parsed name, and the Address column contains the parsed e-mail address. Nice and clean. This information could be added to a database or used in many different ways.

This UDF would be an important part of any ColdFusion-based e-mail application. It's used literally hundreds of times a day in the House of Fusion list archives (www.houseoffusion.com/cf_lists) and runs both fast and smoothly. You can add it to any application using <CFPOP> to allow you to format the results for cleaner display and storage. The only limit to it is that it's a UDF and can be used only with ColdFusion v5 or better. You could remove it from the UDF "wrapper" and add it into older versions of ColdFusion, but versions earlier than CF4 may have some differences in RegEx support.

As you can see from this article, Regular Expressions are extremely powerful even if they look painful. In reality, there are only about two dozen actual commands and they can be picked up easily.

Sources

You can learn more about Regular Expressions at the following locations:

- *Regular Expressions in ColdFusion:* www.houseoffusion.com/RegEx.ppt
- *Regular Expression Power Tips (from Cfun 02):* www.cfconf.org/cfun-02/talks/regularexpressions.ppt and www.cfconf.org/cfun-02/talks/regularexpressionscode.zip
- *Macromedia documentation:* http://livedocs.macromedia.com/cf50docs/CFML_Reference/Functions194.jsp, http://livedocs.macromedia.com/cf50docs/CFML_Reference/Functions195.jsp, and http://livedocs.macromedia.com/cf50docs/Developing_ColdFusion_Applications/regexp.jsp

In addition, a new version of the RegEx bible has been released:

- Friedl, J.E.F., and Oram, A. (ed) (2002). *Mastering Regular Expressions* 2nd ed. O'Reilly.

About the Author

Michael Dinowitz, a founding member of Team Allaire (now Team Macromedia), hosts House of Fusion and Fusion Authority, two well-known ColdFusion resource sites. He also manages the ColdFusion, JRun, and related mailing lists, writes articles and books, trains, and does almost anything else needed to enhance the community. His actual work involves troubleshooting and providing a safety net to programming teams to enhance their effectiveness.

MDINOWIT@HOUSEOFFUSION.COM

Listing 1

```
<!--- Get all headers from the spam account --->
<cfpop action="GETHEADERONLY" name="qSpam" server="houseof-
fusion.com" username="Spam"
password="SpamTrapperTrapsSpam">

<!--- Output the FROM address only --->
<table border="1">
 <tr><td>FROM</td></tr>
<CFOUTPUT query="qSpam">
 <tr><td>#HTMLeditFormat(FROM)#</td></tr>
</CFOUTPUT>
</table>
```

Listing 2

```
 1: <CFSCRIPT>
 2:  function ParseEmail(email)
 3:  {
 4:      // set up local variables for the function
 5:      var truefrom="";
 6:      var address="";
 7:      var name="";
 8:      // Check for the first, second and fifth address
         formats
 9:      var sTrueFrom = REFind (‘^"?([^"<]*)"?
         *<([^@]+@[^>]+)>’, email, 1, 1);
10:      // Check that the RegEx returned a result that we
         can use, if not try a different RegEx
11:      if(sTrueFrom.len[1])
12:      {
13:      // Parse out the name and email portions of the
         address into different variables
14:      name=mid(email, struefrom.pos[2],
         struefrom.len[2]);
15:      address=mid(email, struefrom.pos[3],
         struefrom.len[3]);
16:      }
17:      // If the first RegEx failed than the address must
         either be of the third or fourth formats
18:      else
19:      {
20:      sTrueFrom=REFindNoCase(‘^([-a-z0- 9_.]+@[^
         [:space:]]+) *\(?([^)]+)?\)?’, email, 1, 1);
21:      if (sTrueFrom.len[1])
22:      {
23:      address=mid(email, struefrom.pos[2],
         struefrom.len[2]);
24:      // Check if there was a second sub-expression
         which will contain the name.
25:      if (arraylen(sTrueFrom.len) EQ 3 and
         sTrueFrom.len[3])
26:      name=mid(email, struefrom.pos[3],
         struefrom.len[3]);
27:      }
28:      }
29:      // If the name portion is blank then load it with
         a space
30:      if (Not len(name))
31:      name=" ";
32:      // Return a list containing the name and email
         portions as the result of the UDF
33:      return name&’,’&address;
34: }
35: </CFSCRIPT>
```

Listing 3

```
<!--- Add UDF to parse email address -‡
<CFINCLUDE template="fParseEmail.cfm">
<!--- Get all headers from the spam account --->
<cfpop action="GETHEADERONLY" name="qSpam" server="houseof-
fusion.com" username="Spam"
password="SpamTrapperTrapsSpam">

<!--- Output the FROM address only --->
<table border="1">
 <tr>
  <td>FROM</td>
  <td>Name</td>
  <td>Address</td>
 </tr>
 <CFOUTPUT query="qSpam">
 <!--- set the results of the UDF to a local variable
rather than call it twice. --->
 <CFSET lAddressList=ParseEmail(From)>
 <tr>
  <!--- Remember to use the HTMLEditFormat() function to
display the FROM address. --->
  <td>#HTMLEditFormat(From)#</td>
  <td>#ListFirst(lAddressList)#</td>
  <td>#ListLast(lAddressList)#</td>
 </tr></CFOUTPUT>
</table>
```

CODE LISTING

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

Scale **Models**

The best software? A scale model of the real world

BY **HAL HELMS**

I was telling my son about my childhood the other day. He s 13 now and no longer willing to believe the stories I used to tell him.

Gone are the days when I would recount tales of my pet dinosaur. He no longer finds it credible that rocks had yet to be invented when I was a child. Alas, so young, and already a cynic.

I then began telling him of the many hours I spent making scale models from kits: "We made scale model ships and – best of all – scale model cars! The best of them were intricately detailed, with doors that opened, a hood that revealed a tiny scale model motor, wheels that turned left or right. You had these little plastic pieces that you detached and then carefully glued together. Finally, when all was done, you applied paint and decals and the car was finished."

Later we played a game of *Age of Empires* and I was struck by the similarity between the scale models I used to create and the medieval town that was taking shape in our game. Here was a complete world, filled with buildings, people, and – of course – weapons. Various bands fought each other for possession of territory. It was, in fact, a scale model world.

Such scale models are delightful, and the more complete and faithful they are to their original, the more fun they are to play with. My scale model cars, after all my work, couldn't really drive off anywhere, but here, in a software scale model, the scale model people moved of their own volition and buildings that were attacked burned and were destroyed.

Such play seems worlds away from the task of building software for such mundane activities as managing inventory, selling goods, scheduling tasks and events, and managing people. But is it? The best software is, in fact, a scale model of the real world, a fact obscured by the technical details of building databases and queries and writing code that manipulates this data. But just as the most precise scale models are the most enjoyable to play with, so too are they the most valuable in building software.

Much is made of the notion of "scope creep" – the common occurrence of users adding more and more functionality to a project previously thought to have been well defined. Often, though, such scope creep points out that we haven't taken our play seriously enough, that the scale model worlds we build don't mirror their real-world counterparts closely enough. Users, believing that we have in fact faithfully rendered a software version of their real-world activities, expect that they should be able to do with software what they can do in the real world.

Is that such an unreasonable expectation? Perhaps we developers are too serious in the way we approach building software, not playful enough. Why is *Age of Empires* fun while an Inventory Manager is not? I won't deny that conquering kingdoms is inherently more fun than keeping track of widgets, but I think we can find much greater joy in our work if we remember that we're really building scale models. The problem of scope creep will be far less common if we train ourselves to be faithful model builders.

However, we need good tools for building scale models, tools that help us concentrate more on what the model should be. This is exactly what the goal of object-oriented programming was and is. The first OO language was called, appropriately, Simula. Created in Norway in 1967, the language introduced the idea of classes as blueprints, and objects made from those blueprints. Simula gave way to Smalltalk, Smalltalk to C++, and C++ to Java. All of these provided powerful tools for developers who could use them to faithfully create scale models.

With ColdFusion MX some of that same power is in the hands of ColdFusion developers in the form of ColdFusion components (CFCs). With CFCs we can create scale models of business systems more easily. We can let objects "talk" to one another, making and answering requests from other objects.

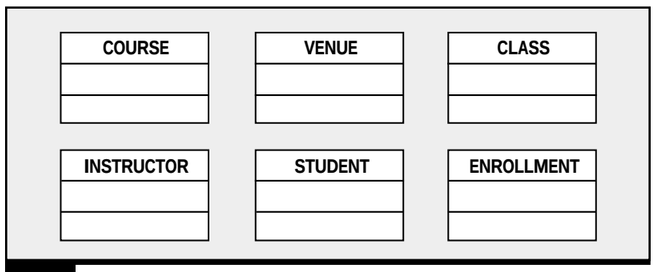

FIGURE 1 Representational boxes in the scale model world

Seeing software in this light lets us change the way we build that software. Rather than start with database entities and relationships, we can think in terms of the actual things themselves. We can rapidly build up a diagram that shows what the inhabitants in our scale model world will be and how they will interact with each other. By trying out scenarios with the diagram, we can determine much more accurately how successful our software will be before we write a line of code.

Let's explore this idea with a small example. We have a client who has a small training company and wants us to write software to help her manage that business. Not surprisingly, the main problems seem to be keeping track of all the classes, students, instructors, and so on. We're committed to building a scale model – and to having fun – so a good start would be to create little boxes, each one representing something distinct that belongs to this scale model world. Each one will at coding time be turned into a CFC (see Figure 1).

We can start trying out scenarios to see if our identification of objects seems to work (see Table 1). You can think of a scenario as a very discrete action that one of the people involved in the system might want to accomplish.

There's a problem with some of our scenarios. Look at the scenario, "A manager wants to see which instructor is qualified to teach a course." It might seem that we could send a message like "isQualified(courseName)" the Instructor class, but that won't work. The Instructor class is only a blueprint for making individual instructors. The same is true for the other classes. Unless we have a way to store a group of the individual objects produced from these classes, we'll have a hard time keeping track of them.

When working through this scale model building exercise, it's helpful to ask, "Who would keep track of that information in the real world?" For example, who would know about instructors, students, classes, and the like? It can't be a particular person, for when he or she leaves, the system's knowledge of those objects will leave too. We can solve this problem by creating a School class that will have the needed groups (instructors, students, etc.) as instance variables.

The more scenarios we work through, and the more we find that our scale model seems to be accurate and complete, the higher our confidence. When we feel we've got a good base to work from, we can begin adding in properties for these classes. You can think of a property as "something the object knows about itself." Information that must be known to the system should be a property of some class. Again, it's helpful to ask, "Who should know this information?"

One advantage of objects over their real-world matches is that they're always completely honest and trustworthy. If you're writing a mortgage application, for example, you can keep the loan balances and other important information with the Borrower. When a Banker needs this information, he can request information from the Borrower, something we decidedly would *not* do in the real world.

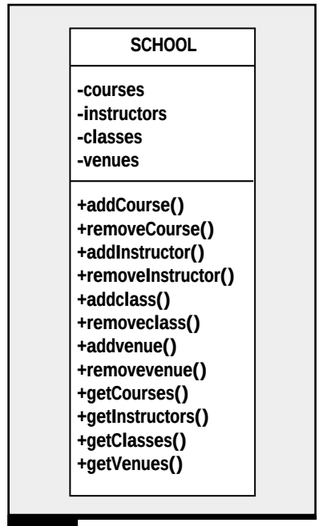

FIGURE 2 Class diagram for the School object

In addition to storing properties – those things the object knows about itself – we must define methods – those requests that the object knows how to deal with. For example, Figure 2 shows the class diagram for the School object.

As we pore over the system, we'll amend it, building up our scale model diagram little by little until, finally, we have a complete system ready to code. And when we are ready for code, CFCs provide an excellent mechanism for ColdFusion programmers to implement objects.

• • •

Building object models, scale models of the real world, is a very different procedure from what most ColdFusion programmers are used to. At first it may seem awkward and unnatural, but for those developers who persevere, the rewards are great: their software is more robust, easier to maintain, and – just as important – way more fun to write!

For more on building object models, take this month's lesson/test at www.halhelms.com.

SCENARIO	OBJECTS INVOLVED
A student wants to take a class in "Java for ColdFusion Programmers"	Student, Class, Enrollment
An instructor wants to see what classes s/he is teaching	Instructor, Class
An instructor wants to see who has signed up for a class	Instructor, Class, Student, Enrollment
A manager wants to see which instructor is qualified to teach a course	Instructor, Course
A student wants to see where a class will be held	Course, Class, Venue
A student wants to see if his/her class has been paid for	Student, Class, Enrollment

TABLE 1 Example scenarios

ABOUT THE AUTHOR

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

@ HAL@FUSEBOX.ORG

Building Applications for **Fun and Profit**

BY
JASON CLARK
AND
DOMINIC PLOUFFE

Got a good idea? Design, program and test

How often have you worked on a project or piece of code and thought, This is something that could save someone a lot of time, or This is something that solves a business problem ?

If this thought crossed your mind, there's a good chance that someone would pay for your application. There's a diverse market out there to buy ColdFusion applications. However, ensuring that your product is up to the task can be challenging, but it's worth the effort.

Building and selling ColdFusion applications requires you to look at your application from different angles. There's a lot to think about when designing an application: how it will be administered, what platforms it will run on, what database servers will be supported, what languages will be supported, how easily you can maintain it once distributed, and what kind of load it will take. In this article we're going to cover some of our experiences in building a packaged ColdFusion application.

Administering the Application

One of the most important components of an application is its administration. Not every application requires an administration section, but most do. For every feature or option in your application, there should be a corresponding setting that can be enabled or disabled, or simply adjusted. End users desire full control over the application. To design a settings structure quite easily, you can use one of several methods, such as XML files, database, and flat files.

Putting your application settings in a database is probably the easiest and most flexible of the three. During the life of your application, new settings will be incorporated and old ones removed. Releasing product upgrades becomes a simple task. To accomplish this, create a table for your settings and include a small code snippet in your application.cfm. Listing 1 contains an example.

Now that a settings structure has been established, create an administration interface so that users can update and change their settings. It's unfortunate, yet very common, for developers not to spend enough time designing an administration interface. Take our advice – invest in an interface that's attractive, intuitive, and familiar (e.g., similar to the standard Windows interface). The less time you have to devote to administrating the product, the better off you'll be. Frames present a very intuitive and flexible interface, if they're designed appropriately.

User Authentication

User authentication can be one of the most important elements of a Web-based application. The public section of the product may not require any login; the administration module will, however. You should spend a significant amount of time planning the authentication module, since it's the gateway to the security of the application. Try to design an authentication module that will allow clients to use their own existing environment. Many organizations use LDAP, NT Authentication, or numerous other sophisticated security systems, and an open authentication module should satisfy their needs. They'll be able to integrate the application easily into their Web site.

Since the Web is stateless, you'll need to decide which method to use in order to recall the user once he or she has logged in. It's wise to make this decision early in the application design process. There are several ways to keep a state on your application, including sessions and cookies. Keeping in mind that there's no perfect technique, the following points should be taken into consideration. Cookies are fast and efficient, and, most important, they don't use any server resources. If the target market permits their use, it's certainly the ideal technique to use.

As the application's popularity increases, the opportunity to crack the design of the authentication module will also increase. Consequently, test the application properly before releasing it to the public. Any stored data on a user's computer should be encrypted using a key that is unique to each client. The ColdFusion Encrypt and Decrypt functions should be sufficient to secure the data.

Listing 2 is a basic example of an authentication design using cookies.

Hosting Considerations

When you create a Web application for clients whose Web site is hosted on a public server, there are certain limitations. Hosted clients have special needs due to the lack of control over their environment. In your planning stages, to avoid future problems, measure these needs carefully.

One limitation to be receptive to when planning for hosting providers is that they can restrict the use of some ColdFusion tags, which might limit the features you had in mind for the application. However, if you must include a feature that requires a restricted tag, don't remove it. Simply ensure that your system requirements are well documented. The most frequently restricted tags are CFDIRECTORY, CFFILE, CFCONTENT, CFEXECUTE, CFOBJECT, and CFREGISTRY. A complete list of the tags that may be restricted by hosting providers can be found in the security section of the ColdFusion administration module.

Last, research various popular hosting providers to ensure that your application will work on their servers. Hosting providers usually have similar guidelines that their clients need to follow.

Modularizing Code

There are a number of good published articles about the benefits of modularizing your Web site and the tools that Macromedia provides. Cfincludes, custom tags, and functions are aways available. Since the release of ColdFusion MX, using ColdFusion components can make the code easier to read, faster, and reusable.

Identify sections of the application that are often repeated and modularize them. If the application is designed for use with ColdFusion 4.x or 5.x, cfincludes are recommended over custom tags. Although the latter might be better at modularizing code structurally, you could encounter problems when using them, such as a decrease in performance. Furthermore, clients may not be able to access the custom tag directory, although you could code your application to use CFMODULE and possibly ColdFusion mappings to help mitigate that problem. Cfincludes are easy to use and don't reduce the performance of the application, and there are no extra system requirements.

Your application will also gain from a well modularized code base in aspects that aren't present when designing a Web site. The modularization will ensure that multiple database systems can be used easily by your application. The more database platforms that are available, the better it will be for the application. In order to be competitive, a Web application should have as few restrictions as possible placed on the client. Most database platforms are similar, and nearly all vendors offer development versions of their products so the database modules can be tested.

If stored procedures can't be used for queries, consider using cfincludes. Cfincludes will allow the creation of a separate template for queries that are different from database platform to platform. Inserts and selects are queries that frequently have platform differences.

Performance

Perhaps the most important aspect of the product is performance. Unfortunately, it's also one of the greatest challenges of product design. Sometimes you have to think a little differently than you would normally. While writing the code, keep in mind the performance with respect to speed. The goal is to have clean, readable code, of course, but that isn't always possible.

Let's say you have a table called "clients." In this table are individual records on each client. Within your application you have some sort of security system that determines the permissions of each client. Typically, this is done by the record ID of the resource, so a client's access rights might be 1,2,5,7. Now, conventional database design would dictate that you should have a separate table with the client's record ID in it along with the resource ID. But

in some cases this will cause overhead that you don't need or want. Alternatively, you could create a field in the client table and store the access rights as a string. Sure, there are DBAs out there who will cringe at the idea, but performance is critical, and sometimes that's just the way it has to be done.

To read more about performance aspects of ColdFusion and its applications, take a look at "ColdFusion Performance and Beyond," Jason Clark's feature article in the August 2002 issue of ***CFDJ***.

Packaging

You've made the application. Now you need to package it. Create an installer so your prospective clients can purchase and download it. The installer is crucial. It should be easy to use, and it should do as much of the installation as possible. Many applications require some sort of database access. By using CFQUERY, you can set up the entire database right from your install scripts – it's that simple. You can create the tables, stored procedures, and indexes from some straightforward ColdFusion scripts (see the example in Listing 3).

The ability to create tables and indexes is common to most database management systems, though specific syntax may vary. Some of these examples use Microsoft SQL Server syntax. If your application runs on multiple database platforms, read the database platforms' documentation to get the syntax for creating the various parts of the database. Note also that the ability to use these CREATE statements in SQL may be limited by the datasource definition and its "CF Settings." For more information about using these sorts of DDL statements and related issues, see Charlie Arehart's article, "Reconfiguring Remote Databases via SQL," in the July 2000 *CFDJ* at www.sys-con.com/coldfusion/article.cfm?id=131.

There are various approaches to packaging an application, such as using InstallShield-type packaging applications or others. These aren't inexpensive or necessarily easy to use. For the most part, a ColdFusion application contains just code and the necessary scripts to create the database. The important thing to consider when creating your installer is the method by which users are going to get the installer to their environment. A significant number of people will either be hosted or may not have access to the file structure or physical machine on which your application is being installed.

Our recommendation is to zip or tar your application, and distribute it as such. The client can simply unzip or untar it and FTP it to its destination. As long as your install scripts do most of the work, they'll only need to create a datasource (or have their hosting provider create their datasource), run the install, and edit an INI file that tells your application the datasource information.

Getting the Word Out

Now that a good foundation has been established for building a Web application, all that remains is a good idea. Take some time to design, program, and test before you release your application to the public. It's advantageous if you can find your first client early in the design process. That client would likely have a specific need for your application and would be a good beta tester – and with a little luck may even pay your development costs! One final suggestion: advertise your application on ColdFusion tag galleries, Macromedia's Developer Exchange (http://devex.macromedia.com/developer/gallery/), and popular search engines.

ABOUT THE AUTHORS

Jason Clark, cofounder and CTO of Fusetalk Inc., has more than 10 years of experience in programming specifically and the IT sector in general.

Dominic Plouffe, cofounder and VP R&D of FuseTalk Inc., has been designing Web applications since 1997.

@ JASON@FUSETALK.COM

@ DOMINIC@FUSETALK.COM

Listing 1: Application.cfm example

```
<!-- Retrieve application settings -->
<cfquery name="qGetSettings" datasource="#ds#"
username="#dsusername#" password="#dspassword#" cachedwith-
in="#createtimespan(0,0,30,0)#">
 select settingname,settingvalue from appsettings
</cfquery>

<!-- Loop through database settings and use SetVariable to
dynamically create the setting name and values -->
<cfloop query="qGetSettings">
 <cfset APP_TEMPVAR = SetVariable(settingname, Trim(set
  tingvalue)>
</cfloop>

Code Example:
<!-- Using one of the dynamically created settings here we
are determining if we will show the users username or his
first and last name -->
<cfif APP_NAMEFORMAT eq "U">
      <cfoutput>#Username#</cfoutput>
<cfelse>
    <cfoutput>#Firstname# #LastName#</cfoutput>
</cfif>
```

Listing 2: Login.cfm page

```
 <cfif USERLOGGEDIN eq "No">
  <cfif isDefined("action")>
    <cfif action eq "loginuser">
<!-- if a user is trying to log in, validate the information
    posted from the form with the users table in your
    database -->
    <cfquery name="qValidateUser" datasource="#ds#">
          select userid,username from myusers where
          username = '#username#'
          and password = '#password#'
 </cfquery>

 <!-- If the user has been validated -->
 <cfif qValidateUser.recordcount neq "0">
   <cfset key = "abcdef"> <!--Your clients should be able
   to set this themselves -->
   <!-- encrypt the value that will be inserted in the
   cookie-->
   <cfset userid = Encrypt(qValidateUser.userid, key)>

   <!-- create the cookie -->
   <cfcookie name="authcookie" value="#userid#">

   <!-- redirect the user to the index page -->
   <script>self.location = 'index.cfm';</script>
 <cfelse>
   <!-- if the user was not validated display error mes
   sage -->
   The username and password that you have entered is
   incorrect.
 </cfif>
</cfif>
 </cfif>
```



```
  <!-- Display the form to enable users to log in -->
  <form action="login.cfm" method="post">
    Username: <input type="text" name="username"><br>
 Password: <input type="password" name="password"><br>
 <input type="submit" value="login">
 <input type="hidden" name="action" value="loginuser">
  </form>
</cfif>

Application.cfm

<cfset USERLOGGEDIN = "No">

<cfif CGI.SCRIPT_NAME does not contain "login.cfm">
  <!-- If the user is not logged in, redirect them to the
  login template -->
  <cfif IsDefined("cookie.authcookie") eq "False">
    <cflocation url="login.cfm">
  <cfelse>
    <cfset key = "abcdef"> <!--Your clients should be able
    to set this themselves -->

 <!-- decrypt the value of the cookie -->
    <cfset userid = Decrypt(cookie.authcookie, key)>

 <!-- validate the user with the decrypted value -->
    <cfquery name="qValidateUser" datasource="#ds#">
      select userid,username from myusers
      where iuserid = #userid#
    </cfquery>

    <cfif qCheck.recordcount eq "0">
   <!-- If the user was not validated redirect them to the
   login page -->
      <cflocation url="login.cfm">
    <cfelse>
   <!-- If the user was validated, set the login flag and
   any other user settings needed (i.e. username, e-mail
   address, color settings -->
      <cfset USERLOGGEDIN = "Yes">
      <cfset USERNAME = qValidateUser.username>
    </cfif>
  </cfif>
</cfif>
```

Listing 3: Installation script example

```
<!-- Create Table -->
<cfquery name="qCreate" datasource="fusetalk"
username="#username#" password="#password#">
create table app_users (
iuserid int not null,
dtinsertdate datetime not null)
</cfquery>

<!-- Create Index -->
<cfquery name="qCreate" datasource="fusetalk"
username="#username#" password="#password#">
CREATE  CLUSTERED  INDEX idx_userid ON app_users(iuserid)
</cfquery>

<!-- Create Stored Procedure -->
<cfquery name="qCreate" datasource="fusetalk"
username="#username#" password="#password#">
CREATE PROCEDURE app_getuser
@userid int
AS
select iuserid from app_users where iuserid = @userid
</cfquery>
```

CODE LISTING

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

Error Handling in **JavaScript**

BY **STEVE BRYANT**

It s all about detection and prevention

Have you ever been to a site and gotten a pop-up box telling you about a JavaScript error on the page? It can be really annoying.

What's worse is that the person responsible for maintaining the site doesn't even know that the error occurred.

This article isn't about Cold-Fusion error handling. That was covered in 2000 and 2001 in ***CFDJ*** by Charlie Arehart's four-part series, "Toward Better Error Handling," as well as by others. Rather, this article will tell you some ways to trap JavaScript errors on the client. More important, it'll tell you how to communicate these errors to ColdFusion automatically so you can write the errors to a log file or e-mail the errors to yourself and find out about them right away. Indeed, the trick we use may be useful to you in ways beyond just error handling.

Error Prevention

The primary way to prevent JavaScript errors in your application, of course, is to catch them during development. To do this you'll have to know when they occur. Browsers may be set by default to hide them. To correct that you must turn on error display in your browser. This means that you see any JavaScript errors on all sites you visit, but to my mind it's a small price to pay to see them in your own site.

It's essential to know about JavaScript errors on your site, especially when debugging. Before I enabled this in my browsers, I wasted hours trying to figure out the cause of problems that became obvious when I enabled error display in my browser.

To do this in Internet Explorer, go to Tools -> Options -> Advanced and check "Display a notification about every script error". To see errors in Netscape, you must use the JavaScript console. To see the console in Netscape Navigator 4x, you must type "javascript:" into the location window. This will bring up the JavaScript console in your browser window. JavaScript errors will be written to the console as they occur. You can also use the console to test individual JavaScript statements. NN6 allows you to have the JavaScript console display in a separate window by going to Tasks -> Tools -> JavaScript Console.

Property	IE 5	IE 5.5+	NS 6+	ECMA
description	Y	Y	N	N
filename	N	N	Y	N
lineNumber	N	N	Y	N
message	N	Y	Y	Y

TABLE 1 Variables available in selected browsers

Error Trapping with JavaScript

One of the most common ways to trap errors in troublesome ColdFusion code is to wrap a <cftry>/<cfcatch> block around it. If you're not using this method, you really should look into it. Part 4 of Charlie Arehart's series (June 2002) dealt with this topic.

JavaScript actually has try/catch statements of its own. As in CF, this is used to test for errors that you can anticipate. The simple example I use in Listing 1 only shows the error message (called "description" in IE and "message" in NN). You can catch information other than the message/description in that scope as well. However, the other variables also differ from browser to browser. Variables available in one browser aren't necessarily available in another. Table 1 (from *JavaScript: The Complete Reference* by Thomas Powell and Fritz Schneider [Osborne McGraw-Hill]) illustrates some of the variables that will be available in selected browsers.

The main problem with depending on try/catch for error handling in JavaScript is that it was only introduced in IE5 and NN6. This leaves out a good portion of your audience. Additionally, you have to write try/catch statements around every bit of troubling code. While you can nest JavaScript try/catch statements just as you can ColdFusion ones, the nature of JavaScript means that encompassing all of your code in try/catch statements just isn't practical. Besides, we don't have the time to rewrite all of our existing code to use try/catch. There's got to be a better way.

Enter the "onerror" Property

This property of the window object has been available since NN3 and IE4. It's a great way to deal with JavaScript errors on a universal level (sort of like <cferror> – see Part 3 of Charlie Arehart's series [February 2001]). It can be used to deal with any JavaScript errors on the entire page, making it a very powerful error handler. Yet it rarely gets used.

The onerror property is used in the window object and in some cases it's available in other JavaScript objects as well (like the img object). I only cover the onerror property of the window object here.

The onerror property tells the browser to execute a function when a JavaScript error occurs anywhere on the page. If this function returns a value of true, this will also disable the browser's default error handling, including displaying the error message to the user for those who've opted to see them (as above).

Listing 2 is a simple example of the onerror property in use. All this does is display a series of alert boxes when an error occurs. It isn't terribly useful, but does make an effective demo. Note the three parameters used in defining the onerror event handler (function). The information used wasn't explicitly passed to the script, but rather was part of how the browser naturally passes information to the onerror event. Note also that I put it before any other HTML. I did this to demonstrate that you could include this script (or one like it) in your Application.cfm and it would still work. Naturally, you could put any other JavaScript statements in this script.

Bringing ColdFusion into the Picture

Certainly you've gotten information from the browser to ColdFusion before, but for this technique to work smoothly, we must do it in such a way that we don't distract from the user's experience. Notice that no other window is opened, and that our current page is not forced to refresh (which would likely cause the error to recur and put the user in an infinite loop). Any of that could interfere with the user's experience. So how do we do it?

The trick is getting images. Any time you use JavaScript to define an image source, you're making a trip to the server to get that image so it will be ready for use on the page. The source of an image object doesn't have to be an actual image. JavaScript doesn't check what sort of file you're using as the source. So you can call a ColdFusion page via the JavaScript image source functionality. You can even append variables to the URL in your request and JavaScript will dutifully send the request to the page and use that page as the source of an image.

This would cause a problem if you actually tried to display the image, but we have no need to do that. We're just sending information to a ColdFusion page. The image is simply our way to do it without forcing the user to leave the page. This is the only way I know of that this can be done in HTML, that is consistent across browsers, and that doesn't distract from the user's experience.

Once we know this, everything else is just an extension of what we've learned so far. The technique could have other uses as well. For example, you could pull a ColdFusion page that's using <CFCONTENT> to make the page output its contents as a .gif or .jpg instead of HTML. This would allow server-side processing to be used to choose an image without the user having to experience a page reload. While I haven't seen any practical uses for this yet, I'm hoping this suggestion will whet enough appetites that I might hear ideas from others.

Now all you have to do is pass on the variables that were passed to your ColdFusion page as follows:

```
errImage = new Image();
errImage.src = 'JSErrs.cfm?url='
+ escape(errUrl) + '&msg=' +
escape(msg) + '&line=' + line;
```

You can then use this ColdFusion page to send yourself an e-mail or log the error. I generally put a <cfmail> tag in mine and have the JavaScript errors e-mailed to myself. If you make sure that you call a ColdFusion page that's in the same application as the executing script (I always do), you can also send yourself CGI and session variables. Your URL variables will

ABOUT THE AUTHOR

Formerly VP of operations at The Internet Design Firm in Denver, Steve Bryant has since moved to Tulsa, Oklahoma, and now contracts for them as an independent developer. A certified ColdFusion developer, he got his BA in philosophy at Oklahoma State University and still has no idea how that led to a career in Web development.

automatically be included in the URL passed to your ColdFusion page. Listing 3 shows a basic script that could be used as the ColdFusion page to handle the JavaScript error. If you aren't using CF5 or above, you can manually loop through the session and URL variables (as shown in Part 2 of Charlie's series [December 2000]) or you can use the CF_Dump custom tag that should still be available in the Macromedia Developer's Exchange.

All this means that you can be notified about a JavaScript error without the user's even knowing that the error occurred. This makes debugging easier for you and a better experience for your user. Remember to test any code that deals with user input carefully and to use try/catch blocks around it so you can tell your user about these errors. If they're unable to submit a form and don't know why, they'll find it very frustrating.

Summary

In this article I covered some ways to detect and prevent JavaScript errors. I also addressed handling errors in JavaScript. Finally, I covered how to send information about JavaScript errors on your page to ColdFusion and showed how to e-mail those errors to yourself. Hopefully this will help you deal with potential JavaScript errors in your current and future development. Good luck!

@ STEVE@BRYANTWEBCONSULTING.COM

Listing 1

```
try {
   var x == 5;
   //we are just "trying" this because we know it will break
} catch (myerrorname) {
   //IE uses "description", but NN uses "message" instead
   if (myerrorname.description == null) {
      alert("Error:" + exception.message)
   } else {
      alert("Error:" + exception.description)
   }
}
```

Listing 2

```
<script>
<!--
function errAlert(msg, errUrl, line) {
   alert(msg);
   alert(errUrl);
   alert(line);
   return true;
}
window.onerror = errAlert;
// -->
</script>
<html>
<head>
   <title>JS Error Test</title>
<script>
function testJS() {
   fred = barney;
}
</script>
</head>
<body>
<script>testJS()</script>
Loaded
</body>
</html>
```

Listing 3

```
<cfmail
   to="sebtools@yahoo.com"
   from="robot@tidf.com"
   subject="JavaScript Error"
   type="HTML">
There was a JavaScript error. <br>
<br>
CGI.SERVER_NAME = #CGI.SERVER_NAME#<br>
CGI.HTTP_USER_AGENT = #CGI.HTTP_USER_AGENT#<br>
<br>
<cfdump var="#Session#">
<br>
<cfdump var="#URL#">
</cfmail>
```

CODE LISTING

The code listing for this article is also located at

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

Compilation and Precompilation in CFMX Templates Part 1

An updated precompile batch file

BY CHARLES AREHART

Most of you have heard by now that ColdFusion MX templates are compiled into Java. Some may wonder what the big deal is.

Many more of you may be surprised to learn that you can precompile CFMX templates. Do you know why you might want to? And if you've known previously about how to precompile CFMX templates, did you find that trying to do so with code outside the default \coldfusionmx\wwwroot directory (or \inetpub\wwwroot for IIS users) was troublesome?

I've got the solution to that problem as well. First, some background to bring everyone up to speed on the subject of compilation and precompilation in CFMX.

CFMX Templates Are Compiled

One of the great new benefits of ColdFusion MX is that, because it's built on an underlying Java platform, our ColdFusion templates are actually compiled into Java classes. They're not really executables, though, and they're not self-sufficient programs that can run on any Java (or even J2EE) platform, but the compilation takes place nonetheless.

Unlike in Java and many other languages, however, this compilation takes place automatically. Just as with previous releases of ColdFusion, you can create or edit your templates, and ColdFusion simply runs them as soon as they're changed. (Actually, JSP templates in the J2EE world work the same way.)

You needn't really concern yourself with the mechanisms of automatic compilation. It works without your doing anything. So why this article?

The Onetime Cost of Compilation to a User

Each of these features, the automatic compilation and the automatic detection of code changes, is a two-edged sword. There's the cost borne by the first user to execute a page after it's been created or edited. The request for that Web page will be delayed for a few seconds while CFMX compiles the code the first time. Some find this cause for concern.

The benefit of compiling, of course, is that once this "cost" has been paid by the first user to hit the page, each subsequent page hit is very fast; this is a onetime cost and recurs only if the file is later changed. That compiled code is saved and stored on disk (as a .class file that CF knows how to find and execute), and even if the server is restarted, that previously compiled file will be reused by the server to serve that template. There won't be any more waits for compilation until a developer changes the template again. (Well, almost. More about that in Part 2.)

But what if you deploy a large number of newly created or edited templates in your application? Or what if you frequently change them throughout the day? Then the number of automatic compilations grows. Consider the impact on the first users who hit such changed pages, or the first user hitting your multipage site after you've deployed a number of new templates. That first user may suffer delays on every page as he or she traverses through your site. Ouch.

Anyone who has gone through the ColdFusion Administrator for the first time after installing CFMX will understand the experience. At least that code doesn't change again after CFMX is installed. But you may hit a page some days later that no one else has hit yet, and you'll wait for that compilation then.

First-Time Wait Not Entirely New

Actually, the penalty borne by the first user to run a template after it's created or edited isn't exclusive to CFMX. In CF5 and before, CF also "interpreted" any new or changed CF template, saving the resulting p-code to memory in CF's template cache. But since the process was writing to memory rather than disk, it wasn't quite as noticeable.

Then again, because it wasn't saved to disk, this first-time interpretation had to take place whenever the ColdFusion server was restarted. (The template cache will be discussed further in Part 2.) This need to reinterpret templates on restart led many to complain about the need for real compilation of code to disk.

So compilation is good. But is there anything you can do to avoid the wait by the first executing user? Is this just the price to be paid for the power of automatic compilation?

Precompilation: Avoiding First-User Cost

Part of the answer lies in the question itself. We're relying on the automatic compilation that occurs when the first user runs the template after it's created or edited. What if you could somehow tell ColdFusion to perform that compilation manually? Wouldn't it be useful if there were a command you could issue against your template directory to say "compile all those templates"?

Macromedia didn't provide us with that, and there may be very good reasons (put the conspiracy theories on hold – we're talking about the same compilation that takes place automatically, so protecting intellectual property is not what kept them from offering it).

But a simple three-line batch file has been making the rounds of various mailing lists and blogs (mine included, www.cfmxplus.blogspot.com). The initial version of this batch file seemed to give us the apparent silver bullet we've sought for slaying the first-time delay. The problem was, it only worked for code that was in the default webroot if you followed its original instructions.

I present here a version that does work for CFMX code in any directory. I've also reduced it to just two lines. Simply create a file called precompile.bat with the following lines:

```
set MX_INSTALL=c:\cfusionmx
%MX_INSTALL%\lib\cfusion.jar coldfusion.tools.Compiler
   -webroot %1 -webinf %MX_INSTALL%\wwwroot\WEB-INF %1
```

Just to be clear, there should be two lines in the file: one starting with "set" and one starting with "%MX_INSTALL%".

Replace the first line's c:\cfusionmx value with the location of the CFMX installation. Again, don't worry if your code isn't stored under the wwwroot in that directory. I'll show in a moment that the batch file will accept a parameter that tells it where your code is actually stored, so you can use this over and over to precompile code in different directories.

Save this file as precompile.bat anywhere on your system. Then, from the DOS command prompt, run it from whatever directory you saved the batch file in. If you're not familiar with getting to the DOS command prompt or running .bat files from a specific directory, please seek help from someone with that knowledge.

Now you can execute it, naming the directory whose files (and subdirectories of files) you want to compile, as in:

```
precompile c:\cfusionmx\wwwroot\mydir
```

or

```
precompile d:\inetpub\wwwroot\somedir
```

or

```
precompile d:\myotherdir
```

That third example demonstrates the compilation of code that isn't stored in the normal webroot. To learn how to be able to access such a directory using the built-in CFMX Web server, see the entry at my blog site, http://cfmxplus.blogspot.com/2002_08_11_cfmxplus_archive.html#85347963.

The point is, this version of the batch file will compile any directory of CF templates as long as you specify the complete path to the file (see the next section for other differences between this batch file and the one that was passed around the community a couple of months ago).

Again, note that I said it will compile the files in the named directory and any subdirectories. That's useful if you're expecting it, but annoying if you aren't. I haven't figured out how to keep it from recursing through the subdirectories. If anyone has figured that out, please let me know by e-mail or via the online version of this article at the www.sys-con.com/coldfusion Web site.

If the command is successful, you'll see a display (perhaps a lengthy one) indicating that it's compiling each file, one at a time, and reporting how long it takes. It also reports any syntax errors in the code. Very nifty!

It even repeats the list of errors at the end of the display for easy review. Depending on your operating system, you may be able to scroll up and down in the command window to review the results that have scrolled off the page. Of course, the compiler is smart enough not to try to compile anything other than .cfm and .cfc files.

Note that you could use this to precompile all the files in your entire wwwroot (including subdirectories), using the command:

```
precompile c:\cfusionmx\wwwroot\
```

as an example, assuming that's where the default webroot is, or on IIS:

```
precompile c:\inetpub\wwwroot\
```

Just be aware that this will also ask it to compile such things as the CFDOCS and CFIDE directories that come installed with CFMX. That's over 2,000 files, not counting your own! Use the tool wisely.

And it may be useful to some to be able to compile just a single file, rather than an entire directory and its subdirectories.

You can do that, too, as in:

```
precompile c:\cfusionmx\www-
root\myfile.cfm
```

Finally, I haven't been able to get it to accept a pattern, at least not in this simple two-line version of the batch file.

As an aside, you may wonder if the compiler called by the precompile batch file knows to skip files it's already compiled. It doesn't skip them, but you'll notice that as it reports on each template it finds, the compilation time reported will usually be zero seconds for a file that hasn't changed since the last compile.

Indeed, I've learned from Spike Washburn's blog (thanks to a pointer from Dave Watts) that there's an available "–f" directive to force a compilation of the templates even if they've already been compiled once. One possible use for this is when something seems stuck in the way CF is interpreting a template and you want to force a recompile.

You could modify the script to add a -f argument before the -webroot argument. You could also enable it as a parameter to be passed with a few extra lines of batch file code.

Other Versions of precompile.bat

As I mentioned, the batch file I offer above is slightly different from one that was passed around among beta testers and early users of CFMX a couple of months ago. That version used a value for the –webroot directive on the compiler command line (the "java" line) that caused it not to work for compiling code outside the default webroot.

Some learned that they could change that value to point to their desired directory, but then they learned that in order for it to work, they had to go through some more hoops. What was lacking was two things. First, the critical –webinf compiler directive, a hidden gem I learned from Harry Klein of CONTENS Software in Germany.

As my batch file code above shows, it names the path to the wwwroot\WEB-INF directory where CFMX is installed. Without this, if you pointed the –webroot directive at some location other than the default wwwroot, the compiler would fail because it expected to find that WEB-INF directory and several subdirectories of it under the directory you named in the –webroot directive.

Furthermore, I've set the –webroot directive to be whatever directory you pass on the call to the precompile.bat, rather than pointing it at the default webroot. These two things were the missing pieces that kept the previous versions of this precompile.bat file from working against any directory.

I've also changed it so that you must provide the complete path to the directory containing the code you want to compile (whereas the older one worked with relative paths depending on where you placed the precompile.bat file). This makes mine more robust for a few reasons, at the cost of a few extra keystrokes when passing directory names to the precompile batch file.

Readers familiar with batch file variables will notice that I've also set it up to accept only a single argument naming the directory path or file holding the code to be compiled. The older version allowed specification of multiple directories or files. Again, it worked fine with code in the default webroot, but in order to support code in any directory in just two lines, that feature had to be eliminated.

Indeed, you could change it to accept multiple directories, but with the way the batch file is currently written, they'd have to be subdirectories of the first one named, and that's not useful since it already compiles all the subdirectories of any provided. You couldn't name a parent and only one of its children, for instance, since the naming of the parent would already compile all the children.

One other minor change in this version and others offered previously is that I'm prefixing the call to the java interpreter (in the second line of the batch file) with the location within the CFMX install directory where the default Java Virtual Machine is installed. This might prevent some confusion if multiple JVMs are installed.

With more tweaking and some additional DOS commands and conditional tests, the batch file code could be made to provide additional functionality and address some of these issues. I think most will be glad simply to be able to precompile code at all, in more than just the default webroot, and in just two simple lines of code.

Finally, for those working on Solaris and Linux, ColdFusion developer Matt Liotta has offered a version of that batch file as a shell script:

```
#!/usr/bin/sh

MX_INSTALL=/opt/coldfusionmx
PATH=$MX_INSTALL/runtime/bin:
  $PATH
$MX_INSTALL/jre/bin/java -class
  path\$CLASSPATH:$MX_INSTALL/
  lib/cfusion.jar \coldfusion.
  tools.Compiler -webroot $1\
  -webinf $MX_INSTALL/wwwroot/
  WEB-INF $1
```

Just to be clear, there should be three lines of code in the file, one starting with "MX_INSTALL", one starting with "PATH", and one starting with "$MX_INSTALL/jre/bin/java".

Again, be sure to change line one to the location where CFMX is installed. Thanks, Matt!

• • •

We could stop at this point – and will. The problem of precompiling code is solved. But there will be curious folks (and bit-twiddlers) among you who will want to know more, maybe lots more.

How much time is this really saving? If it's compiled to disk, how and when does CFMX read it into memory to execute it? What's the cost of that? What happens with CFINCLUDEd files? Where does the compiled code go? Can I look at it? Can I just delete the generated class files instead? How do I determine which class file was generated for which CF template? Can I distribute the compiled code on other servers without the source code? (The answer to the last question, sadly, is no.)

We'll discuss that and the rest of these questions next month. Otherwise, you've got all you need to know to start precompiling code in CFMX.

@CAREHART@SYSTEMANAGE.COM

ABOUT THE AUTHOR

Charlie Arehart is a certified Macromedia trainer/developer and CTO of SysteManage. He contributes to several CF resources, is a frequent speaker at user groups throughout the country, and provides training, coaching, and consultation services. He is also CFDJ *s technical coeditor.*

CommonSpot Content Server 3.0 from PaperThin

The best just got better

REVIEWED BY
STEVE DRUCKER
Steve Drucker is the hands-on CEO of Fig Leaf Software, a Macromedia premier solutions and training partner with offices in Washington, DC, and Atlanta.

sdrucker@figleaf.com

VITALS

CommonSpot Content Server 3.0

PaperThin, Inc.

Address:
300 Congress Street
Suite 303
Quincy, MA 02169

Phone: 617 471-4440

Fax: 617 471-4465

Web: www.paperthin.com

Test Environment:
Dell Inspiron 5000, 450MHz P-III, ColdFusion 5.0, 512MB RAM, Windows 2000 SP2, MS SQL Server 7

Pricing: $19,500–$85,000 priced per server and number of content contributors. The `a la carte pricing model allows you to pay only for the features you need. Monthly ASP hosting option available through ISP channel

Currently there are more than 200 commercially available content management systems (CMSs). Each has strengths and weaknesses relative to the others…belying their underlying middleware platform and customer history. While not all are ColdFusion based, I believe that the CommonSpot Content Server 3.0 (the 2001 ***CFDJ*** Readers' Choice winner) from PaperThin, Inc., sets a new standard for flexibility and performance across all middle-tier systems.

Comparing Apples to Apples

Believe it or not, there's quite a bit of controversy within the CMS industry about what functionality a "Web content management" system should bring to the table. After evaluating many CMS applications, however, I've found that most vendors offer the following core set of features:

- Nontechnical content contributors may modify the contents of their Web sites anytime, anywhere, through their Web browser. No additional software is required. Content may range from text articles to uploaded files and images, usually stored in a separate asset repository.
- Page layout is based on a template model. Content is dissociated from formatting, allowing you to change the site layout without affecting data.
- Modified content is subject to an approval process and/or workflow.
- Changes to content are versioned. Prior versions of content may be restored easily.
- Content may be classified through the use of categories and keywords. Conversely, content may be located through scanning metadata or the full text of the article.
- Content may be personalized and scheduled to fit group or individual preferences.
- Reporting allows contributors to determine the popularity of their content. Also, statistics can be generated detailing long page load times, page weight, broken links, recently updated content, and more.
- Sites, sections, pages, and content items can have their access limited through a roles-based security architecture.
- An extensible architecture allows you to link to other databases and applications.

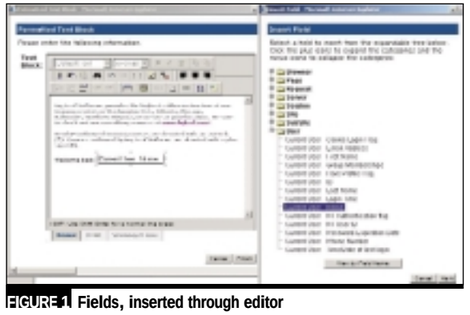

FIGURE 1 Fields, inserted through editor

Who Is PaperThin?

In these troubling times you need to manage your investment risk in software much as you might manage your stock portfolio. The worst-case scenario of a product being discontinued – or worse, the company behind it becoming insolvent – has been a recurring theme throughout this industry. The field is littered with large companies that have consolidated product lines and halted further development as well as small consultancies that opted to design their own custom CMS solutions, only to find that stealing resources from their development business proved untenable.

Fortunately, PaperThin is a closely held, self-financed company that is singularly focused on making CommonSpot the leading CMS available. Perhaps the best testament to the product is that the firm has actually doubled in size and revenue during the last 12 months. At the time of this writing, over 150 CommonSpot-managed sites are currently deployed and they have been successful in building a third-party VAR and ISP channel.

The Devil Is in the Details… or, Rather, the Implementation

Content management systems have fragmented into two camps – "tookit" systems that are very flexible but require a lot of labor to deploy, and out-of-the-box solutions that tend to be somewhat less flexible but require less programming. CommonSpot, thankfully, falls into the latter category. Once an HTML site prototype has been designed, the insertion of two CFML custom tags into the template is often all that's required to enable content creation and editing.

When evaluating CMS you should consider the *total* cost of ownership – the cost of the software and the amount of services required to customize it to your needs. Depending on the package you choose, services costs may vary from one to four times the cost of the product. Based on my experience, CommonSpot deployments typically fall into the low end of this range.

CommonSpot 3.0 Authoring

CommonSpot uses the "anywhere authoring" approach common to many content management systems. Modifying content is as simple as navigating to the page that needs edits, entering "author" mode, and clicking on the appropriate content. A WYSIWYG editor, based on the standard Microsoft Internet Explorer activeX control, allows you to control formatting. CommonSpot administrators may restrict access to each feature of the editor, limiting font, Cascading Style Sheet styles, color choices, and so forth. In its latest release PaperThin has integrated "HTML TIDY," an application that cleans and formats the HTML generated by the Microsoft editor.

Also new to this version is the addition of both internal and external hyperlink verification. Any URLs used within the editor are verified as linking to an existing page. Version 3.0 immediately reports to the author any content that doesn't meet section 508 accessibility standards. New to CommonSpot is the addition of system-variable fields, depicted in Figure 1, that can be inserted directly through the editor and are replaced with dynamic content at runtime.

Taking an Element-Based Approach

CommonSpot's out-of-the-box functionality becomes apparent when you try to place a new data element on a page. As depicted in Figure 2, the system supports over 50 different data types through an element "gallery." These include items commonly found on sites, such as breadcrumb navigation, pop-up DHTML menus, data entry forms, and formatted text blocks. The system will also automatically convert Microsoft Word documents and PowerPoint presentations to HTML. Form-based wizards step content contributors through adding content to each element.

CommonSpot 3.0 now allows integrators to define custom elements specifying the form that a user submits data through, and element display templates, which govern the format of the information when published as HTML.

Templates That Act Like Transparencies

CommonSpot uses a unique "layering" approach to the template publishing metaphor. Each page is built from a series of templates that function like virtual transparencies. Each template may contain unique content, formatting, layout, and security information. Any page based on that "virtual" template inherits changes to any "layer."

Workflow That You Can Use

Many higher-end systems allow you to graphically define multistep workflows through venn diagrams. However, in my experience most of these use cases ultimately fail under their own weight and complexity. CommonSpot's workflow definition system is simple by comparison, yet very effective. An unlimited number of approval levels may be defined within the system. Each level is typically composed of specific users and/or groups. When content is submitted, managers at each level are notified via e-mail when it's their turn to sign off on the content.

At each point, a manager may approve the content, refer the content back to the author for editing, reject the content modification completely, or bypass approval for subordinate approvers if the manager him/herself is making the change. Therefore, if a content modification needs to be published immediately, a high-ranking approver may be authorized to come into the system, bypass the other approvers, and publish the change immediately.

Integration with FuseTalk

An integral part of many sites are discussion forums where members may post messages and comment about issues of the day. CommonSpot now interfaces directly with the award-winning FuseTalk forums from FuseTalk, Inc. (www.fusetalk.com). Adding an interactive discussion to any CommonSpot Web page can now be accomplished in seconds, allowing you to post information and have your readers carry on an interactive discussion about the content – if you dare!

Scaling Your Deployment and Licensing

Like many CMS vendors, PaperThin licenses their product based on the number of content contributors within your organization (the number of end users who can view the content is limited only by your particular hardware configuration). Their newest release allows you to purchase either "dedicated" or "shared" content contributor licenses. Under the former model, you can designate a fixed number of content contributor accounts. Under the "shared" license, however, you may designate an unlimited number of contributors but are limited to a number of concurrent logins. Increasing your license is as simple as copying a license file into a specific directory on the server.

Any good CMS provides facilities for scalability and high availability. The new release now allows you to scale your site easily over multiple machines through content replication. Essentially, content contributors use a single CommonSpot server to effect their changes. The authoring server then syndicates those changes on a scheduled basis to any number of deployment servers for public view.

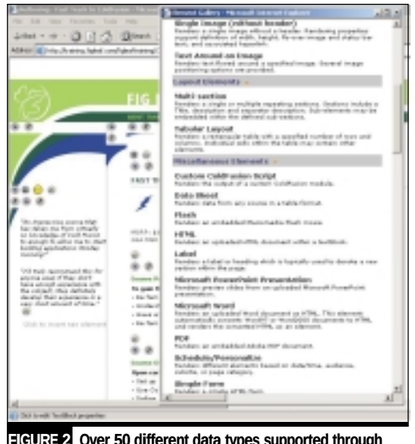

FIGURE 2 Over 50 different data types supported through element "gallery"

PRODUCT SNAPSHOT

Target Audience:
Any organization that has to manage a lot of text-based content. Traditional verticals include trade associations, state and federal government, universities, life sciences, and intranets.

Pros:
- Simple to deploy
- Scalable architecture
- Easy for nontechnical content contributors to master
- Rich set of built-in elements
- Great support for CSS and Section 508
- Easily extensible through CFML

Cons:
- Limited support for PDA / device output / XML / XSLT

Client Platform: Content contributors must use IE 5.X+ / PC

Server OS: NT4 server with IIS and CF4.5

Database Support: MS SQL Server, Oracle, MS Access

Using Integers to Store **Bits of Information**

Simplify your forms and database design

BY **TOM NUNAMAKER**

Imagine that your client, Fast Eddy s Auto World, asks you to build a data entry form for his inventory.

There are several models of cars that use combinations of many options. No two use the same combination. How can you efficiently design the form and database to display the correct options and store the information in a database without constantly adding and deleting fields in your data table? The answer is to use integers and bitwise operators.

We first need to understand how integers are stored in binary format. Our society teaches base 10 to manipulate numbers. We are so familiar with it that we sometimes forget that behind its familiarity lies a system for expressing numbers by using columns representing different values. For example, the number 12 looks different when the columns are exposed:

1000	100	10	1
0	0	1	2

That can be read as "zero thousands + zero hundreds + one ten + two ones".

Computers use base 2 (so you have two conditions: on and off). That same number 12 decimal expressed as binary numbers would be:

8	4	2	1
1	1	0	0

This can be read as "one eight + one four + zero twos + zero ones".

Binary numbers have made terrific flags for years since each place is either ON (1) or OFF (0). If you view the 1100 binary number as a series of four flags left to right, we could say:

The left-most flag is ON
The second flag is ON
The third flag is OFF
The fourth flag is OFF

Fast Eddy's form doesn't have flags. It has automobile options. If you assigned names to each flag, you could also say:

The Power windows flag is ON
The Automatic Transmission flag is ON
The Tilt Steering Wheel flag is OFF
The Leather Seats flag is OFF

Let's design two database tables to hold these bits of information (see Figure 1). These tables are tied only to each other and to your form. They aren't connected with referential integrity to your inventory table. Your integer storage fields store the sum of user-chosen bitValues.

In the *BitGroups* table we'll store the names of each group of information. BitGroups help you organize your collection of bits. In the *Bits* table we'll store the specifics of each group. Each combination of bitgroup_id and bit is unique. For instance, Fast Eddy might have these BitGroups and bitValues:

BitGroups

BitGroup_ID	BitGroup
1	Interior Options
2	Exterior Options

Bits

BitGroup_ID	BitValue	BitTitle
1	0	Leather Seats
1	1	Cloth Seats
1	2	Power Seats
1	3	CD Player
1	4	AM/FM Radio
2	0	Power Windows
2	1	Rear Spoiler
2	2	Fog Lights
2	3	Chrome Rims
2	4	Heated Rear View Mirrors

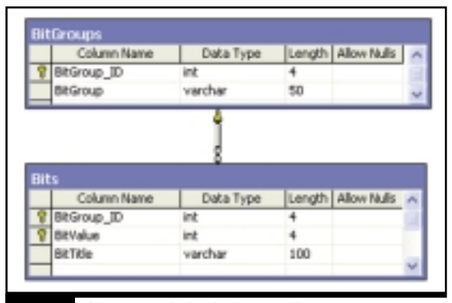

FIGURE 1 BitGroups and Bits database tables

To store these bits of information in Fast Eddy's VehicleInventory table, you'd only need two fields, InteriorOptions and ExteriorOptions. If either of these fields is zero, then none of that BitGroup's options are checked (all of the flags are zero). If any combination is checked, you add the bitValues raised to the power of 2. That's why we start with a bitValue of zero: 2 to the zero power is 1, or our first flag position. You can store the powers of 2 in the database Bits table but I prefer not to have ColdFusion do the math for me.

Typical 32-bit integers, like those found in SQL Server, can handle 31 bits of information. This is the maximum number of bits one integer can store. To store more than that, you have to use multiple storage integers. In Fast Eddy's case we can divide his options into logical groups such as Interior and Exterior Options. If you further break each of these groups into two, you can display these options in two columns on your HTML form. Most forms won't require 31 fields in each of two columns down the page. Keeping your integers storing fewer than 31 bits of information leaves future expansion possibilities. If Fast Eddy's option lists expand so much that you have to exceed 31 bits, you'll have to split your options into smaller groups.

Let's display the checkboxes for Fast Eddy's automobiles. We could generate the powers of 2 now, but if the checkboxes aren't checked, it's a bit of wasted processing. Let's do the binary conversion after the form is submitted. Here are our checkboxes:

```
<input type="checkbox" name="interiorOptions_0"
value="1">Leather Seats<br>
<input type="checkbox" name="interiorOptions_1
value="1">Cloth Seats  <br>
<input type="checkbox" name="interiorOptions_2"
value="1">Power Seats  <br>
<input type="checkbox" name="interiorOptions_3"
value="1">CD player     <br>
<input type="checkbox" name="interiorOptions_4"
value="1">AM/FM radio  <br>
```

The ColdFusion code to generate those checkboxes would be:

```
<cfoutput query="getBits1">
  <input type="checkbox"
         name="interiorOptions_#bitValue#"
         value="1">
  #bitTitle#<br>
</cfoutput>
```

After the user selects the appropriate options, the receiving page has to add up the values before inserting or updating the vehicleInventory table. Since we haven't converted the choices to powers of 2, we'll check to see if the checkbox exists and calculate the bitValue here. This is the code to add up the checkbox values:

```
<cfset interiorOptionsTotal = 0>
<cfloop index="i" from="0" to="4">
  <cfif IsDefined("form.interiorOptions_#i#")>
    <cfset interiorOptionsTotal = interiorOptionsTotal +
2 ^ i>
  </cfif>
</cfloop>
```

Use the interiorOptionsTotal variable to place into your record. If no checkboxes were selected, the value is zero. If any were selected, their values are added to create a unique combined number. For instance, if Power Seats (value of 2 ^ 2 = 4) and CD Player (value of 2 ^ 3 = 8) were selected, their sum (12 decimal or 1100 binary) would be inserted into the InteriorOptions field for that vehicle. Note that no other combination of options adds up to 12. It is *always* this exact combination.

When we edit the form with existing data, we have to figure out which bits were checked when we display the form. Here's a

Fast Eddy needs a different form for each model in his inventory. You're thinking what a nightmare that is..."

revised version of our code to generate the checked="checked" in the proper checkboxes. getDetails is the query containing our vehicleInventory data:

```
<cfoutput query="getBits1">
  <input type="checkbox"
         name="interiorOptions_#bitValue#"
         value="1"
         <cfif IsNumeric(getDetails.interiorOptions)
           AND BitAnd(getDetails.interiorOptions, 2 ^
bitValue) GT 0>
           checked="checked"
         </cfif>>#bitTitle#<br>
</cfoutput>
```

BitAnd function compares the particular bit to the value in the database. If the database flag is set, bitwise AND will be true (GT 0) and checked="checked" will be displayed in the form.

IsNumeric is used to check for NULL values. If you don't allow NULLS and define a default value of zero, the IsNumeric() function isn't needed.

The preceding code will generate this HTML, assuming interiorOptions is set to 12 (01100 binary):

```
<input type="checkbox" name="interiorOptions_0"
value="1">Leather Seats  <br>
<input type="checkbox" name="interiorOptions_1"
value="1">Cloth Seats    <br>
<input type="checkbox" name="interiorOptions_2"
value="1"
                    checked="checked">Power
Seats    <br>
<input type="checkbox" name="interiorOptions_3"
value="1"
               checked="checked">CD player
<br>
<input type="checkbox" name="interiorOptions_4"
value="1">AM/FM radio    <br>
```

So "Power Seats" and "CD Player" are selected as we'd expect. Fast Eddy is impressed. He wonders how you're going to handle different models of automobiles, though. He needs a different form for each model in his inventory. You're thinking what a maintenance nightmare it is to maintain that many similar forms. One change has to be replicated in many places. Unless there's a way to exclude particular fields on a particular form. You can build Fast Eddy's forms from one form template by associating the model with a list of options that model doesn't have. If Fast Eddy's cars were wildly different, you could approach it by including only the designated options. We'll assume that most cars are about the same and deal with excluding the different options from each model.

Let's modify the VehicleModels table and add a field called BitExcludeValue with a default value of zero (see Figure 2).

VehicleModels

	Column Name	Data Type	Length	Allow Nulls
🔑	VehicleModel_ID	int	4	
	BitExcludeValue	int	4	

FIGURE 2 VehicleModels table

If you want to exclude a bit from one vehicleModel, set the BitExcludeValue to the sum of all bitValues for the particular form that Fast Eddy wants to exclude. For instance, to exclude "Power Seats" (value 2 ^ 2 = 4) and "AM/FM Radio" (value 2 ^ 4 = 16):

Name	Value
Power Seats	2 ^ 2 = 4
AM/FM Radio	2 ^ 4 = 16
BitExcludeValue	= 20

Fast Eddy would store VehicleModel information for each model that includes the BitExcludeValue for that model. When you display the form for that model, you'd query the database to retrieve the BitExcludeValue and use it in the form display code like this:

```
<cfoutput query="getBits1">
  <cfif BitAnd(getDetails.BitExcludeValue, 2 ^
bitValue) IS 0>
<input type="checkbox"

name="interiorOptions_#bitValue#"
                value="1"
                <cfif
IsNumeric(getDetails.interiorOptions)
                    AND
BitAnd(getDetails.interiorOptions, 2 ^ bitValue)
GT 0>
             checked="checked"
             </cfif>>#bitTitle#<br>
  </cfif>
</cfoutput>
```

BitAnd returns logical TRUE when both bits are TRUE (1). If the b itValue matches the BitExcludeValue, they're both TRUE and BitAnd would be TRUE (1) so we won't display the checkbox for that option. If the BitExcludeValue is zero, BitAnd will be FALSE (0) and we'll display the checkbox for that option.

When we display a form with the BitExcludeValue of 20 (10100 binary), we see this:

```
<input type="checkbox" name="interiorOptions_0"
value="1">Leather Seats  <br>
<input type="checkbox" name="interiorOptions_1"
value="1">Cloth Seats    <br>
<input type="checkbox" name="interiorOptions_3"
value="1">CD player      <br>
```

Checkboxes in HTML are passed only if they're checked. They are *not* passed if they aren't checked or if they don't exist on the form. BitValues of 2 and 4 will never appear on our action form since they don't exist on this form with this exclude combination. We're safe in using this form with any combination of checkboxes on any of Fast Eddy's forms.

When Fast Eddy tells you a new option is available, just add it to the bits table. You don't have to redesign your database tables by adding or deleting fields as options change over time. If an option changes its name (Leather Seats changes to Corinthian Leather Seats), just update your Bits table.

You aren't limited to storing form checkboxes with this type of database storage method, of course. Hal Helms's bitwise security model uses this same idea to store user permissions. It's a great way to store a large number of flags in a compact space.

ABOUT THE AUTHOR

Tom Nunamaker, a USAF T-37 instructor pilot, has been programming in various languages since 1974. He started ColdFusion programming in 1996 and is a certified advanced CF5 developer. His work includes integrating pilot flight plans to the European Air Traffic Control System for the USAF and Web hosting. He has posted several custom tags at www.toshop.com.

@ TOM@TOSHOP.COM

Extending ColdFusion

Welcome to the first edition of a new *CFDJ* column

BY
RAYMOND CAMDEN

Raymond Camden is a software engineer for Macromedia, Inc. A longtime ColdFusion user, Raymond is a former member of Team Macromedia and a contributor to the "Mastering ColdFusion" series published by Sybex. He also presents at numerous conferences and contributes to online webzines and CFDJ. He and Rob Brooks-Bilson created and run the Common Function Library Project (www.cflib.org), an open source repository of ColdFusion UDFs. Raymond formed and helps manage the Hampton Roads ColdFusion User Group (www.hrcfug.org).

ray@camdenfamily.com

This column will focus on some of the ways developers have of extending the core capacities of ColdFusion. Specifically, we will cover user-defined functions (UDFs), custom tags, and ColdFusion components (CFCs). Every other month I'll select one or two resources that are freely available to anyone and explain what makes it (or them) interesting – or just plain cool. If you'd like to suggest a particular resource, please e-mail me at ray@camdenfamily.com.

I'll begin this month with a pair of UDFs developed by **Nathan Dintenfas** (http://nathan.dintenfass.com). The UDFs are named QueryString ChangeVar (www.cflib.org/udf.cfm?id=471) and QueryStringDeleteVar (www.cflib.org/udf.cfm?ID=472). You can probably guess by their names what they accomplish. Both UDFs work on the query string. Not sure what that is? The query string is everything you see in the URL after the location of the file. In the following URL:

```
http://www.cflib.org/udf.cfm?ID=472
```

the file name is udf.cfm. Everything after the question mark (?) is the query string. Most likely you've worked with the query string in the past and not realized it. Whenever you run a ColdFusion file that has a query string passed to it, the ColdFusion server automatically converts the query string into URL variables. Using the example above, the URL variable "ID" would be created with a value of 472.

There are times, however, when you want to use the current query string, and modify it. Let's consider a few examples. If you've ever written a self-posting form, then you probably had code that looked like this:

```
<form action="edit.cfm" method="post">
```

In this case the form will post back to itself – a file called edit.cfm. What happens if you come back later and decide that "edit.cfm" isn't a very descriptive name? If you change the filename to edit Article.cfm, you have to remember to update the form tag as well. One way around this is to simply make the action value of the form tag point to the current page dynamically:

```
<form action="#cgi.script_name#"
   method="post">
```

This code will work even after you rename the file. Taking things a bit further, your form may have used certain URL variables. For example, an edit form may have used url.id to determine which record in the database to load. Your form tag would need to pass that as well. As before, you could hard-code it, but it's just as easy to use the query string cgi variable:

```
<form action="#cgi.script_name#?#cgi.query_
   string#" method="post'>
```

What's nice about this method is that it will work with any filename, and will automatically pass back any URL variables that were sent to the file originally. This method works great…up to the point where your query string contains a large set of values – one of which you need to remove. Consider the following query string:

```
sort=title&sortDir=down&filter=bugs&start=21
```

This query string contains four name/value pairs. Three of them are used by an application to retrieve and sort data. The fourth item, "start", is used to determine what record to begin displaying. As you can imagine, this query string is typical of "next-*n*" type data interfaces. Where things get tricky is when you want to grab the query string, but without the start value. This is because you'll need to set a new start value. You could rebuild the query string from scratch. If you know that the string should contain url.sort, url.sortDir, and url.filter, this isn't a big deal. However, if the URL parameters change in the future, this is one more place where if code isn't updated, the application will break.

By using either of the two UDFs, Query StringChangeVar or QueryStringDeleteVar, we can easily modify the query string. Here's an example:

```
<cfoutput>
<a href="#cgi.script_name#?#queryString
   ChangeVar("start",newstart)#">Next 20
   Results</a>
</cfoutput>
```

This example uses the QueryStringChangeVar UDF to change the value of url.start to newstart (a value defined earlier.) The UDF returns the entire modified query string, so we simply add it after the question mark.

How does the UDF work? It's really not that complex. Listing 1 contains the code for the UDF. We'll focus on the important aspects of the function, after the initial var statements. The query string (which can either default to cgi.query_string or use a string passed in) will always be in the form of name=value&name2=value2; we can treat it as a list. The UDF uses the listToArray function, treating the ampersand (&) as the list delimiter. This will return an array in which each element can be treated as a list as well as in the form of "name=value". We loop through each element of the array and grab the name portion. If it matches the value we want to change, we use the new value passed in. While we loop, we build up a new query string value that the UDF will return. Last but not least, if for some reason the value we want to change wasn't provided, we add it to the end. (QueryStringDeleteVar works in a similar fashion and can be found in Listing 2.)

• • •

That's it for this edition. As I said earlier, please send me your suggestions for UDFs, custom tags, or CFCs to highlight.

Listing 1

```
<cfscript>
/**
 * Changes a var in a query string.
 *
 * @param name    The name of the name/value pair you want
   to modify. (Required)
 * @param value   The new value for the name/value pair you
   want to modify. (Required)
 * @param qs      Query string to modify. Defaults to
   CGI.QUERY_STRING. (Optional)
 * @return Returns a string.
 * @author Nathan Dintenfass (nathan@changemedia.com)
 * @version 2, September 5, 2002
 */
function QueryStringChangeVar(variable,value){
 //var to hold the final string
   var string = "";
 //vars for use in the loop, so we don't have to evaluate
   lists and arrays more than once
 var ii = 1;
 var thisVar = "";
 var thisIndex = "";
 var array = "";
 var changedIt = 0;
 //if there is a third argument, use that as the query
   string, otherwise default to cgi.query_string
 var qs = cgi.query_string;
 if(arrayLen(arguments) GT 2)
  qs = arguments[3];

 //put the query string into an array for easier looping
   array = listToArray(qs,"&");
 //now, loop over the array and rebuild the string
   for(ii = 1; ii lte arrayLen(array); ii = ii + 1){
  thisIndex = array[ii];
  thisVar = listFirst(thisIndex,"=");
  //if this is the var, edit it to the value, otherwise,
  just append
  if(thisVar is variable){
   string = listAppend(string,thisVar & "=" & value,"&");
   changedIt = 1;
  }
  else{
   string = listAppend(string,thisIndex,"&");
  }
 }
 //if it was not changed, add it!
   if(NOT changedIt)
  string = listAppend(string,variable & "=" & value,"&");
 //return the string
   return string;
}
</cfscript>
```

Listing 2

```
<cfscript>
/**
 * Deletes a var from a query string.
 * Idea for multiple args from Michael Stephenson
   (michael.stephenson@adtran.com)
 *
 * @param variable        A variable, or a list of
   variables, to delete from the query string.
 * @param qs      Query string to modify. Defaults to
   CGI.QUERY_STRING.
 * @return Returns a string.
 * @author Nathan Dintenfass (nathan@changemedia.com)
 * @version 1.1, February 24, 2002
 */
function queryStringDeleteVar(variable){
 //var to hold the final string
 var string = "";
 //vars for use in the loop, so we don't have to evaluate
   lists and arrays more than once
 var ii = 1;
 var thisVar = "";
 var thisIndex = "";
 var array = "";
 //if there is a second argument, use that as the query
   string, otherwise default to cgi.query_string
 var qs = cgi.query_string;
 if(arrayLen(arguments) GT 1)
  qs = arguments[2];
 //put the query string into an array for easier looping
   array = listToArray(qs,"&");
 //now, loop over the array and rebuild the string
   for(ii = 1; ii lte arrayLen(array); ii = ii + 1){
  thisIndex = array[ii];
  thisVar = listFirst(thisIndex,"=");
  //if this is the var, edit it to the value, otherwise,
    just append
  if(not listFind(variable,thisVar))
   string = listAppend(string,thisIndex,"&");
 }
 //return the string
 return string;
}
</cfscript>
```

CODE LISTING

The code listing for this article is also located at

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

CF USER GROUPS

Alabama
Birmingham, AL CFUG
www.bcfug.org

Alabama
Huntsville, AL CFUG
www.nacfug.com

Alaska
Anchorage, AK CFUG
www.akcfug.org

Arizona
Phoenix, AZ CFUG
www.azcfug.com

Arizona
Tucson, AZ CFUG
www.tucsoncfug.org

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

Northern Colorado
Northern Colorado CFUG
www.nccfug.com

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

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/

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

Florida
Tallahassee, FL CFUG
www.tcfug.com/

Florida
Tampa, FL CFUG
www.tbcfug.org

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

Atlanta
Georgia CFUG
www.cfugorama.com

Georgia
Columbus, GA CFUG
www.vcfug.org

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

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/

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

Massachusetts
Boston, MA CFUG
www.cfugboston.org

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

Missouri
Kansas City, MO CFUG
www.kcfusion.org

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

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

Nebraska
Omaha, NE CFUG
www.necfug.com

Nevada
Las Vegas, NV CFUG
www.sncfug.com

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

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

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

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

Ohio, Columbus
Ohio Area CFUG
www.oacfug.org

Ohio
Mid Ohio Valley MMUG
www.movcfug.org/

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

Oklahoma
Tulsa, OK MMUG
www.tulsacfug.org

Oregon
Eugene, OR CFUG
www.EugeneCFUG.org

Oregon
Portland, OR CFUG
www.pdxcfug.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

Rhode Island
Providence, RI CFUG
www.ricfug.com

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

Tennessee
Nashville, TN CFUG
www.ncfug.org

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

Texas
Dallas, TX CFUG
www.dfwcfug.org/

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

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

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

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

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

Wyoming, Jackson
Wyoming MMUG
www.wycfug.org

International

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

Belgium, Brussels
Belgium CFUG
www.cfug.be

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

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

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/

Malaysia, kuala lumpur
Malaysia CFUG
www.coldfusioneer.com/

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/

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

Switzerland
Martin Bürlimann
Switzerland CFUG
www.cfug.ch

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

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

Turkey, Izmer
Turkey CFUG
www.cftr.net

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

United Kingdom, London
UK CFUG
www.ukcfug.org

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

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

CF Community

Tales from the **List**

SQL can be your friend

BY **SIMON HORWITH**

This installment of "Tales from the List" focuses on SQL much more than ColdFusion. Databases and the SQL used to manipulate their data account for roughly 85% of all ColdFusion application performance issues. Let's examine a thread from the **CFDJList** that deals with this topic in the more specific context of a list member's application.

Evik James, who was having SQL problems, began the thread. He stated that he has a page that runs a two-table join query that results in having to execute processor-intensive code in order to calculate the total values in one table as they relate to unique entries and a total in the other. The page was loading very slowly compared to other pages in his application. His query looked like:

```
SELECT    U.UserID, U.Name,
COUNT(L.QStatusID) AS TotalQ
FROM    Users U LEFT JOIN Leads L ON
U.UserID = L.UserID
WHERE  L.QStatusID = 4 AND
GROUP BY U.UserID, U.Name,
ORDER BY U.Name
```

and resulted in:

```
User    TotalQ
Jose     42
Mary     10
Bob       5
```

Unfortunately, Evik wanted to get results back in the following format:

```
UserName, TotalQ, TotalUnQ
Jose       42       37
Mary       11       19
Mark       15       65
```

Note that a value of 5 rather than 4 is used to compute the TotalUnQ column value. **I-Lin Kuo**, a frequent list contributor, suggested CASE statements, which results in the following fix:

```
SELECT    U.UserID, U.Name,
 SUM( CASE WHEN L.QStatusID =4 then 1
else 0 END) as TotalQ,
 SUM( CASE WHEN L.QStatusID =5 then 1
else 0 END) as TotalUnQ
FROM    Users U LEFT JOIN Leads L ON
U.UserID = L.UserID
WHERE  L.QStatusID in (4,5) AND
GROUP BY U.UserID, U.Name,
ORDER BY U.Name
```

Jolly Green Giant, another list contributor, asked whether this could be done in Access. **Jason Gulledge**, unsure of how to do this in Access, then chimed in and reminded everyone that on Oracle databases you'd want to look into using the "DECODE" function. **Christopher Dempsey**, responding to the question regarding this functionality in Access, stated that in Access you can use the IIF function to accomplish the same result. The entire thread was followed up by a post from **Ray Camden** that it never ceases to amaze him how much cool stuff you can do in SQL. I'll leave you with his final statement, which sums up the point of all this:

"As a general tip – if you ever find yourself doing a query and then doing postprocessing before displaying – double-check what you are doing – it may be possible in SQL instead."

@ SHORWITH@FIGLEAF.COM

ABOUT THE AUTHOR

Simon Horwith, a senior developer and Macromedia-certified ColdFusion instructor at Fig Leaf Software in Washington, DC, has been using ColdFusion since version 1.5. He is a contributing author to Professional ColdFusion 5.0 *(WROX) as well as technical editor of* The ColdFusion 5.0 Certification Study Guide *(Syngress).*

October **2002**

What's Online

www.sys-con.com/coldfusion

ColdFusion Online

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

Readers' Choice Awards

Go to www.sys-con.com/coldfusion and vote for your favorite custom tag, database tool, software, development tool, or book in the **2002 ColdFusion Developer's Journal Readers' Choice Awards**.

Awards will be presented for the products that receive the most votes in each category. Winners will be honored at Macromedia DevCon in October.

These are the only ColdFusion awards given in the industry, and they are given to products nominated and selected by the largest number of professional ColdFusion developers ever assembled for such a purpose.

Write for Us

Do you have something you want to write about the ColdFusion world? We're interested in real-world implementations, industry analysis, and tutorials. Check out the author guidelines at www.sys-con.com/coldfusion/writers.cfm and send us a proposal.

Answers Are a Click Away

Are you blocked by your blocking factor? Are your synchronization objects out of sync? Visit the ***CFDJ*** Technical Forum (www.sys-con.com/fusetalk/categories.cfm?catid=6) and see what other developers have to say. You can ask a question, share a tip, improve your code, find a way to solve a glitch, and network with ColdFusion professionals. Recent postings have generated online discussions on security issues, data importing, dialog boxes, submit buttons, image info tags, ColdFusion forms, and more.

Subscribe to Our Free Weekly Newsletters

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

Product Reviews

Considering a product upgrade? Want to know the ins and outs of a new product before you purchase it? Make sure you read our in-depth product reviews.

Our writers get behind the hype and give you the facts. All products are tested by experts and leaders in the information technology industry.

CFDJnews

Macromedia Announces Three New Versions of CFMX

(*San Francisco*) – Macromedia, Inc. has announced the availability of Macromedia ColdFusion MX for J2EE Application Servers. Macromedia ColdFusion MX, the rapid server scripting environment for creating rich Internet applications, brings the proven ease of use and productivity of ColdFusion to the highly scalable, standards-based Java technology architecture.

Versions of Macromedia ColdFusion MX are specifically optimized for IBM WebSphere Application Server, Sun ONE, and Macromedia JRun. Each version is available for immediate download from the Macromedia Online Store at www.macromedia.com/go/buycfmx_j2ee/.

ColdFusion MX is built on a breakthrough new architecture that delivers the scalability, reliability, and power of the Java platform without the complexity.

Customers can leverage the rapid development capabilities of the ColdFusion scripting environment to accelerate their development projects while deploying their applications either on a standalone ColdFusion server or on top of leading Java application servers.

Macromedia ColdFusion MX for IBM WebSphere Application Server is also available for immediate download from IBM's Web site at www-3.ibm.com/software/webservers/coldfusionmx/.

Macromedia and IBM jointly developed the product and will work together through joint sales, marketing, and technical support activities. IBM is the first third-party J2EE platform provider company to resell a version of ColdFusion MX. ColdFusion MX for IBM WebSphere Application Server is available directly from Macromedia and for purchase via IBM's Passport Advantage.

Macromedia and Sun Microsystems collaborated on the development of ColdFusion MX for Sun ONE, and both companies will jointly execute sales and marketing activities. As a result, ColdFusion developers and Java 2 Platform, Enterprise Edition (J2EE) developers can use ColdFusion MX for Sun ONE to share Java objects, which will dramatically increase development productivity and lower costs across IT organizations. Additionally, Web application developers without Java programming skills can easily leverage ColdFusion MX to productively build and deploy applications on the Sun ONE platform.

Macromedia MX Developer Resource Kit Now Available

(*San Francisco*) – Macromedia MX Developer Resource Kit, Volume One, is now available from Macromedia, Inc. The kit, the first of a quarterly series, contains extensions, components, and resources to enable Macromedia Flash MX, Dreamweaver MX, and ColdFusion MX developers to quickly and easily create and deploy rich Internet applications. Macromedia MX Developer Resource Kit, Volume One, costs $99 and is available for immediate purchase from the Macromedia Online Store at www.macromedia.com/go/buydrk/.

The kit contains cross-product articles and resources, Macromedia Flash MX components, and Dreamweaver MX extensions. These extensions and components for Macromedia Flash MX and Dreamweaver MX are only available as part of this kit, which also contains an archive of more than 400MB of content from the Macromedia Designer & Developer Center, including multiple sample applications, tutorials, and other articles. A detailed list of the kit contents is available at www.macromedia.com/go/drk/.

Web Content Management Solution Supports Microsoft ASP.NET and PHP

(*Amherst, NH*) – Ektron, Inc., a producer of developer-friendly Web content authoring, publishing, and management technologies for non-technical end users, has released Ektron CMS100 version 2.0. In addition to supporting Microsoft ASP and Macromedia ColdFusion, the new release supports Microsoft ASP.NET and PHP platforms.

Ektron's browser-based content management solution provides core content authoring and publishing capabilities, and costs $499. Ektron CMS100 offers a familiar word processor–like editing toolbar and intuitive interface for content publishing by nontechnical users. Web professionals can easily configure and customize the solution to maintain control over navigation, look and feel, and other site infrastructure.

Web-Native Software Model Enables Rapid Product Innovation Without Customer Disruption

(*San Francisco*) – Atomz, a provider of enterprise Web site management software delivered as an online service, has announced numerous enhancements to its flagship Atomz Publish and Atomz Search applications.

The enhancements are the latest in a constant stream of improvements the company makes to increase the capabilities of its enterprise software products. Recently the company announced the Atomz VPN Solution, a secure virtual private network offering that complements its already robust and secure application infrastructure, and WebDAV, which supports its Web content management product, Atomz Publish.

CFDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
ACTIVE PDF	WWW.ACTIVEPDF.COM	949.582.9002	4
CF ADVISOR	WWW.CFADVISOR.COM	201.802.3069	49
CFDYNAMICS	WWW.CFDYNAMICS.COM	866.CFDYNAMICS	19
CFXHOSTING	WWW.CFXHOSTING.COM	866.CFX.HOST	27
E-ZONE MEDIA	WWW.FUSETALK.COM	866.477.7542	9
HOSTMYSITE.COM	WWW.HOSTMYSITE.COM	877.215.HOST	31
INTERLAND	WWW.INTERLAND.COM	800.845.8706	23
INTERMEDIA	WWW.INTERMEDIA.NET	800.379.7729	52
JDJ STORE	WWW.JDJSTORE.COM	201.802.3069	33, 37
MACROMEDIA	WWW.MACROMEDIA.COM/GO/CFMXAD	877.460.8679	11
MACROMEDIA	WWW.MACROMEDIA.COM/GO/CFDJDEVCON2002	877.460.8679	13
MACROMEDIA	WWW.MACROMEDIA.COM/GO/CFMXLIGHT	877.460.8679	15
NEW ATLANTA	WWW.NEWATLANTA.COM		2, 3
ON-LINE DATA SOLUTIONS, INC.	www.coolfusion.com	631.737.4668	25
PACIFIC ONLINE	WWW.PACONLINE.NET	877.503.9870	35
PAPERTHIN	WWW.PAPERTHIN.COM	800.940.3087	29
SAMS PUBLISHING	WWW.SAMSPUBLISHING.COM		51
SYS-CON MEDIA	WWW.SYS-CON.COM/SUBOFFER.CFM	201.802.3069	33
WEB SERVICES EDGE WEST	WWW.SYS-CON.COM	201.802.3069	43

General Conditions: The Publisher reserves the right to refuse any advertising not meeting the standards that are set to protect the high editorial quality of. All advertising is subject to approval by the Publisher. The Publisher assumes no liability for any costs or damages incurred if for any reason the Publisher fails to publish an advertisement. In no event shall the Publisher be liable for any costs or damages in excess of the cost of the advertisement as a result of a mistake in the advertisement or for any other reason. The Advertiser is fully responsible for all financial liability and terms of the contract executed by the agents or agencies who are acting on behalf of the Advertiser. Conditions set in this document (except the rates) are subject to change by the Publisher without notice. No conditions other than those set forth in this "General Conditions Document" shall be binding upon the Publisher. Advertisers (and their agencies) are fully responsible for the content of their advertisements printed in ColdFusion Developer's Journal. Advertisements are to be printed at the discretion of the Publisher. This discretion includes the positioning of the advertisement, except for "preferred positions" described in the rate table. Cancellations and changes to advertisements must be made in writing before the closing date. "Publisher" in this "General Conditions Document" refers to SYS-CON Publications, Inc.