CF'S NEW BUILDING BLOCKS Pg.14

U.S. $8.99 (Canada $9.99)

ColdFusion Developer's Journal

ColdFusionJournal.com

July 2002 Volume: 4 Issue: 7

SYS-CON MEDIA

by Ronald West
page 6

COLDFUSION Developer's Journal

editorial advisory board
Jeremy Allaire, *CTO*, **macromedia, inc.**
Charles Arehart, *CTO*, **systemanage**
Michael Dinowitz, **house of fusion, fusion authority**
Steve Drucker, *CEO*, **fig leaf software**
Ben Forta, *products*, **macromedia**
Hal Helms, *training*, **team macromedia**
Kevin Lynch, *chief software architect*, **macromedia**
Karl Moss, *principal software developer*, **macromedia**
Michael Smith, *president*, **teratech**
Bruce Van Horn, *president*, **netsite dynamics, LLC**

editorial
editor-in-chief
Robert Diamond robert@sys-con.com
editorial director
Jeremy Geelan jeremy@sys-con.com
executive editor
M'lou Pinkham mpinkham@sys-con.com
managing editor
Cheryl Van Sise cheryl@sys-con.com
editor
Nancy Valentine nancy@sys-con.com
associate editors
Jamie Matusow jamie@sys-con.com
Gail Schultz gail@sys-con.com
Jean Cassidy jean@sys-con.com
assistant editor
Jennifer Stilley jennifer@sys-con.com
editorial intern
Stephanie Williams stephanie@sys-con.com

production
vp, production & design
Jim Morgan jim@sys-con.com
lead designer
Cathryn Burak cathyb@sys-con.com
art director
Alex Botero alex@sys-con.com
associate art directors
Louis F. Cuffari louis@sys-con.com
Richard Silverberg richards@sys-con.com
Aarathi Venkataraman aarathi@sys-con.com
assistant art director
Tami Beatty tami@sys-con.com

contributors to this issue
Ben Forta, David Gassner, Hal Helms, Simon Horwith, Joshua Oster-Morris, Tom Peer, Jeff Peters, Neil Roberts, Kevin Schmidt, Ronald West

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

The Latest Big Thing…

BY **ROBERT DIAMOND**

I took a vacation a few weeks ago, traveling to Los Angeles for a couple of meetings (can never get totally away!) and some fun under the sun. I arrived at the Orange County airport about 20 minutes ahead of schedule and phoned the friend who was going to pick me up. Her husband answered and informed me that she'd forgotten her cell phone so he had no way to contact her, but the plan called for her to run by their house and then to pick me up. To make a long story less long, she then called him from a pay phone to say she was skipping the house and coming straight for me; he called me and promised to pick me up before dark if she didn't show. Sixty minutes later she showed up to get me, so the story had a happy ending (until the later sunburn, of course).

Now, why did I tell you that story? For two reasons, actually: the first, to strongly advocate time management (and picking people up on time!). Task lists and deadlines are a regular part of large-scale development projects. Many of us working on smaller projects don't bother with them, seeing them as a corner to cut. Most people who have been developing for years will tell you that it's definitely worth doing as a "best practice."

There are lots of great CF-based task list project managers out there that are worth using, and they'll definitely make your life easier in the long run. Macromedia's custom tag library is probably the best place to start if you're looking for one. Especially for group development projects – I've always found them to be a big help, and I'm sure you will too.

As for the second reason, during an unplanned hour of free time, I caught up on some reading about Web services. They're certainly the latest "big thing" in the *i*-technology world – both their introduction and adoption. Many CFers used XML as the "diving board" to jump into these new, uncharted waters. We've been covering XML for a long time now, and, as I'm sure regular readers have noticed, have slowly expanded our Web services coverage in reaction to reader demand. This month, ***CFDJ*** is focusing on Web services – attempting to chart those waters for you.

On that note, a slightly shameless plug: for those interested in entering the worlds of Web services or XML full-scale, **SYS-CON Media**, *CFDJ*'s wonderful parent, publishes great magazines on those too. (Full information at www.sys-con.com.)

• • •

We've just begun nominations for the 2002 **ColdFusion Developer's Journal Readers' Choice Awards**. This year we'll have all of the great categories from last year: *Best Book, Best Consulting Service, Best Custom Tag, Best Database Tool, Best Design Service, Best e-Business Software, Best Education and Training, Best Testing Tool, Best Web Development Tool, Best Web Hosting, Best Web Site, Best Web Application*, and finally, *Most Innovative CF Application*.

We've added a few new categories, too – for Web services, Java, and Flash integration products, and services recognizing ColdFusion's continued growth and expansion.

Nominations will be accepted through July 31 (www.sys-con.com/coldfusion). Voting begins on August 1 and will run through September 30. We'll be recognizing and announcing the winners this fall at **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 Ronald West

CFMX & Web Services

Connect your CF app to the world

Over the next few months ColdFusion developers across the globe will be handed the keys to a next-generation Internet application development model. The culmination of over four years of work, which began even before the current release of ColdFusion went into development, it will generate a revolutionary way to develop and share application data and business logic over the Internet.

For many who have been involved with CF since the early years, this model will be welcomed, and those of you who are just about to embark on your CF tour of duty are in luck. The ColdFusion MX (CFMX) development architecture will make developing and maintaining CF applications a whole lot easier. In addition, Macromedia has enriched the CF toolset to include the easiest procedures to deploy and invoke Web services.

This article:

- Discusses the importance of Web services and addresses the necessary steps to set up and use Web services within a CF application
- Explains how to create a Web service and outlines the functionality of the cfcomponent, cffunction, cfargument, and cfreturn tags
- Discusses the cfinvoke and cfinvokeargument tags and explains how to consume a Web service through the use of CFScript (a Web service can be invoked through an HTML form and a Flash client as well)

Fundamentals of a Web Service

In the past it may have been difficult for "pure" CF developers to stay on top of the hoopla that surrounds Web services, let alone participate in a deployment project. Although there's been some integration available with JRun, the tools for such a developer simply didn't exist. It's important to understand the fundamentals of a Web service, because they'll play a large role in our future application development environment. Since I'm sure most of you are familiar with Web services, I'll keep the description brief.

A Web service is specific code that exists on a network that is made available for use by other connected systems. To invoke a Web service you essentially call a method written on the remote system from a client. The method is code written in a particular language exposed through the combination of a SOAP engine and a transport engine. The SOAP engine takes input/output parameters and serializes/deserializes them into a particular XML format, SOAP (Simple Object Access Protocol). The transport engine sends the data over the Internet from its place of origin (a client) to its destination (a server). The most basic and probably most popular transport engine will be HTTP. The transport type, however, doesn't limit a Web service.

A Web service is registered with the SOAP engine through description files. The engine is tightly integrated with the application server. Once the request arrives at the SOAP engine via the transport engine, the request is parsed and the code is invoked. The return data is then encapsulated into a SOAP response by the SOAP engine and sent back to the transport engine where it is relayed to the client that made the original request (see Figure 1). All of the method's input and output parameters, as well as the Web service code, can be accessed through a Web browser in the WSDL (Web Services Description Language) file. WSDL files are also written in XML and can be easily read. The actual format of the WSDL file is outside the scope of this article, but it's important to understand the mechanism used to describe the Web service.

SOAP engines currently exist for many different platforms written in several different languages. Since the CFMX runtime engine is now J2EE compliant, and Apache has produced a SOAP engine for Java called AXIS, the two applications together create the foundation for Web services support in ColdFusion. Simply put, CFMX will provide the fastest way to deploy and consume Web services. CFMX has taken out a lot of the configuration details required to expose code through a Web service. It takes care of the registration required between your code and the SOAP engine and offers multiple options for the consumption of Web services written in many different languages.

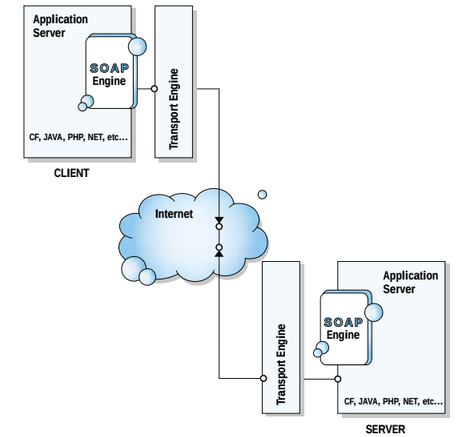


Figure 1 Dynamics of a Web service

Could It Be Any Simpler?

Methods (or functions) weren't available to CF developers until CF5 with the introduction of user-defined functions. UDFs allow for concisely written reusable code to be made available throughout an application. Internally, this is a great way to write reusable code. Unfortunately, the code isn't accessible from any external location, and the only way it can be distributed is if the source code is revealed.

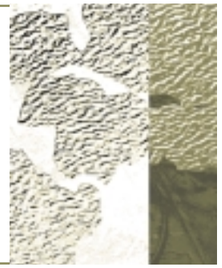
CFMX introduces components. From a ColdFusion Component (CFC) a Web service can be built instantly. In fact, there's little difference between the architectures for a component and a Web service. (The difference is discussed later, but for now they can be considered one and the same.)

The code example in Listing 1 is a UDF from the example apps in CF5. The UDF converts degrees Celsius to degrees Fahrenheit and vice versa. In CF5 the code was included as a common library and could be accessed with code like:

```
#TempConvert(temp, scale)#
```

The UDF had two input parameters: ATemp (the temperature to be converted) and ItsScale (the scale for the given temperature). The parameters are passed to the function in sequential order. The return data is a simple string, which represents the temperature converted into the new scale. Again, this is a fantastic way to write and access reusable code within an application; however, a UDF can't be distributed easily.

> **"From a CF Component a Web service can be built instantly. In fact, there's little difference between the architectures for a component and a Web service"**



A few changes are needed to convert this code to a Web service, allowing you to distribute the functionality of the code easily and not lose control over the source (see Listing 2):

1. Add the cfcomponent tag around the entire document. The tag lets the CF application server know how to process the code.
2. Next, replace the Function line (and end line) from inside the cfscript tag with a cffunction tag that surrounds the entire cfscript tag.
3. Remove the return lines and replace them with cfreturn tags outside the cfscript tag.
4. Once required arguments are defined with cfargument tags, change the file name from cfm to cfc and you're finished.

Except for the movement of code and a file rename, not much has changed on the code side, but a lot has changed on the application side. You've just created your first Web service – it's that simple! No description files, no registration, no server reboot. Now CF can register a method automatically with the built-in SOAP engine. Just a few lines of code, a simple rename of the file, and voilà, an instant Web service. The CFMX server understands how to expose the method so other applications written in other languages can access your code securely.

The Tags

The cfcomponent tag has a few optional attributes that add to its extensibility:

- ***hint:*** Use this to add metadata that describes your Web service.
- ***extends:*** Add this attribute and you essentially borrow code from another CFC as if you explicitly called the other component with a cfinclude.
- ***output:*** Add this attribute to automatically handle the output of #expr# tokens within cffunction tags.

The cffunction tag has one required attribute:

- ***name:*** This attribute registers the method name. In our example the name of the function is tempconvert. This is the method that will be invoked remotely.

In addition, cffunction has *optional* attributes, as described here:

- ***returnType:*** Use this when you return data to the client. The type is either any, array, binary, boolean, date, guid, numeric, query, string, struct, uuid, variableName, void (this option doesn't return a value), return. In the tempconvert example the return type was a string.
- ***roles:*** This adds a comma-separated list of CF security roles allowed to access this function.
- ***access:*** The level of access defined for the function:

—*private:* Makes the function available to the component that calls this method
—*package:* Allows the function to be made available to other components in a package (a package can be created as a collection of components)
—*public:* Makes the function available to local requests only (i.e., not to any locations other than the local site)
—*remote:* Makes the function available as a Web service; allows the function to be called both locally and remotely

The cfargument tag has only one required attribute:

- ***name:*** Refers to the name of the input parameter sent by the caller page.

Additionally, the cfargument tag has some optional attributes that add to its functionality:

- ***type:*** This attribute represents the expected data type of the argument. Valid values are any, array, binary, boolean, date, guid, numeric, query, string, struct, uuid, variableName.
- ***required:*** Either true or false – if false, an additional attribute can be specified to define a "default" value for the argument.

The example had two arguments, "ATemp" and "ItsScale"; both were of the type string and both were required. So the example now has two cfargument tags, which define those two arguments (parameters) as required. The cfargument tag also tells users that the data type for these tags is of datatype string.

The cfreturn tag acts just like the return line of the cfscript tag in our original UDF. As only one cfreturn tag can be present in a cffunction tag, only one value can be returned to the client that calls the page. However, different data types can be returned, so to return a set of data, just create a structure with the data and use cfreturn to send it back to the client.

(Security and access are two very important parts of a Web service. For many Web services you'll want to restrict access entirely or just to certain portions of your Web service. The security aspect of Web services is discussed briefly later on.)

Using the New Web Service Internally

Earlier I mentioned that in CFMX a Web service is merely a component…with a slight difference. The only real difference between a component (code available only to an internal CF application) and a Web service (CF code available on remote systems through a SOAP call) is the attribute ACCESS in the cffunction tag.

Set the ACCESS attribute to REMOTE and you designate the code as a Web service. Set it to any other value and you designate the code as a CFC.

To invoke a Web service we must introduce a new tag, cfinvoke. Don't despair, though. If you like to use CFScript there's also a set of tools to call the Web service from within a CFScript tag – I'll discuss that as well. In addition, we need to talk a little more about the WSDL file (remember, this is the file formatted in XML that describes our Web service).

When Web services first began, the WSDL file had to be produced by hand. As Web services technology advanced, tools were written on certain platforms that could analyze a Web service and create a WSDL file for you. In CFMX neither of these processes is necessary; in fact, CFMX simply creates a WSDL file on the fly. To view the WSDL for a Web service created in CFMX, merely point an XML-compatible browser at the Web service page and add the WSDL query string to the end (i.e., http://localhost:8500/cfdj/com/tempconvert.cfc?WSDL). The browser returns the WSDL file automatically.

Another great feature of CF Web services is that CFMX supports introspection. The CFMX server will display all of the methods available for all Web services written in CF. In addition, the CFMX server will display all of the input/output parameters and their data types for each method offered in the Web service. Currently, access to this functionality is available only to users who know the RDS password for the CFMX server. Check the CFMX documentation for more on introspection.

Although comparable, a little more code is involved when you use the cfinvoke tag in place of a simple function call such as you have with a UDF. This tag offers a common methodology for connectivity to Web services, both internally and externally. The code required to connect to the TempConvert Web service (see Listing 3) allows for an interface into the methods of a Web service. Because our Web service has required parameters (arguments), you can see that nested inside the cfinvoke tag are two cfinvokeargument tags, which take data from the caller page to be passed to the Web service.

The cfinvoke tag has a ton of options and you'll use the tag in different ways. It all depends on what your code invokes. For Web services we use:

- ***webservice:*** This attribute value is the URL location for the WSDL file (discussed earlier). In addition, you can define a map to your Web service in the CF Admin under "Web Services." All you do then is place the Web service name here.
- ***method:*** The name of the method (code enclosed in cffunction tag) goes here.
- ***returnVariable:*** With this attribute you can name the return variable (data returned in the cfreturn tag). If you place "myVar" in here, you can output #myVar# anywhere on the page after the cfinvoke tag.

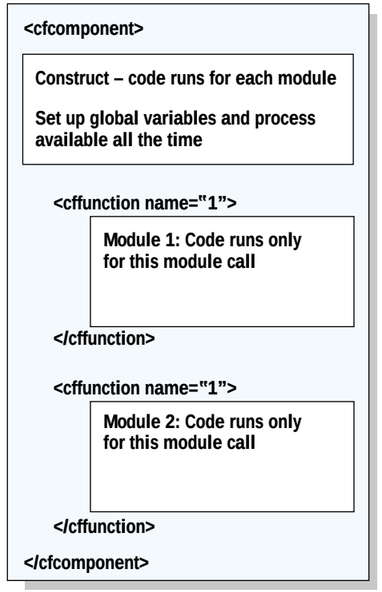


Figure 2 Architecture of the CFMX Web service

You can use multiple cfinvoke tags in the same document.

The Web service can also be consumed in a CFScript tag (see Listing 4). Simply create a Web service object and give it a name (this process will look familiar to anyone who's written script code to access a Java file or COM object). As in the use of the cfinvoke tag, just point to the WSDL file for the Web service (i.e., ws = createObject("webservice", "http://localhost:8500/cfdj/com/tempconvert.cfc?wsdl").

To invoke a method for the Web service, simply use the dot notation with the Web service object name followed by the method name. To pass in arguments to the Web service in a script tag, pass them sequentially, the way they're passed in a UDF, that is, ws.doTempConvert(temp, scale).

If you name the method call, you essentially name the return variable. You can then work with the output. This works the same as the returnVariable attribute in the cfinvoke tag:

```
degs = ws.doTempConvert(temp, scale);
writeOutput("Original Temp:" & temp
  & " " & scale & "<br>");
writeOutput("Converted to: " & degs
  & " F");
```

Important information to keep in mind is that a Web service can consist of any CF code. You can even use cfinclude inside the cffunction tag if you want to keep your code cleaner. Moreover, a Web service has access to all scope variables, which includes Application, Server, Session, and Client.

Beyond Simple Web Services

Our Web service had only one method (cffunction tag) that returned a simple string; ordinarily there would be multiple methods. Each method in a Web service is in charge of a particular task. In a sense, you want to define methods that operate together to perform a set of procedures within a particular domain.

For instance, we could have added another method to our TempConvert Web service, getScale, which would track the scale for the current temperature conversion. If the getScale function were passed "C", it would return "F", and vice versa. Not a crucial function, but it extends the functionality of the Web service a bit. So in our example the TempConvert function would call the getScale function, pass the scale argument, and receive the opposite scale. We could then have combined the data from the getScale function with the converted temperature and returned an array. The temperature data could be stored in the first index and the newly converted temperature in the second index.

Remember, it's possible to return an assortment of data types. It's this basic power of data representation that makes Web services so powerful. Disparate systems now have the ability to share complex data even if the applications are written in different languages. It can be a little tricky to share structures and queries with all languages, but it's possible, and as Web services develop it will become a lot easier.

The CFMX Web service has a unique architecture, which aids in code reuse (see Figure 2). The portion of code (if any) between the cfcomponent tag and the list of methods (cffunction tags) is run every time a method within its body is called. You can refer to the code as the *constructor*. For instance, if you created a Web service that queried a table in the database and you wanted to return three different sets of data to the client – the record count, the query data itself, and the query data – as a WDDX packet, you'd design three methods – getCount, getDataQuery, and getDataWDDX. You'd position the query at the top of the page (within the cfcomponent tag, but before the cffunction tags). All three methods could use the data from the query and return the necessary data to the client. If you use the constructor wisely, it can save a lot of code and be useful when you extend another component.

Since a Web service is essentially CF code, we have all of the same security practices available. You can secure a Web service with Web server authentication, CF application security, role-based security, or programmatic security. While the details for each of these techniques is outside the scope of this article, it's important to understand that all of the current methods for authentication are available to Web services as well.

Where to Go from Here

ColdFusion has entered a new era, and the ability to produce and consume Web services is only the start. What has been introduced in CFMX is a whole new way to communicate over the Web. All of the tools are available to CF developers to show the world what CF can do. Take some time and go through the documentation for CFCs and Web services. Once fully understood, the ideology behind CFCs and Web services should change the way you think about Web application development.

About the Author

Ronald West is a senior applications developer at Paperthin, where he helps maintain the content management application. Additionally, he is a director for the Rhode Island Cold Fusion User Group, where he assists in event planning and overall group direction.

@ RWEST@PAPERTHIN.COM

Listing 1

```
<!---
  The UDF is defined in this block of CFSCRIPT.
  Two arguments are passed to the function, a temperature
and
  the scale that it is in. The temperature is checked to
  make sure it's a number (the value "NAN" is returned and
  processing ends if it is not). Then the calculation is
  processed based on the scale being "F" or "C."   If the
  scale given is neither "F" nor "C," the message "Not a
  valid scale" is returned.
--->

<cfscript>
  function TempConvert(ATemp, ItsScale){
    if (not IsNumeric(ATemp)) return "NAN";

    if (UCase(ItsScale) eq "F")
      // temp given in Fahrenheit. Convert to Celsius
      degs = (ATemp - 32.0) * (5.0/9.0);
    else if (UCase(ItsScale) eq "C")
      // temp given in Celsius. Convert to Fahrenheit
      degs = (ATemp * 9.0/5.0) + 32;
    else
      degs = "Not a valid Scale";
   ;
  return degs;
  }
</cfscript>
```

Listing 2

```
<cfcomponent>

  <cffunction name="doTempConvert" returntype="string"
access="remote" output="false">

   <!--- // two required arguments --->
   <cfargument name="ATemp" required="true" type="string">
   <cfargument name="ItsScale" required="true"
type="string" >

   <cfif not isNumeric(ATemp)>
    <cfset degs = "NAN">
    <cfreturn degs>
   </cfif>

   <cfscript>
   if (UCase(ItsScale) eq "F")
    // temp given in Fahrenheit. Convert to Celsius
    degs = (ATemp - 32.0) * (5.0/9.0);
   else if (UCase(ItsScale) eq "C")
    // temp given in Celsius. Convert to Fahrenheit
    degs = (ATemp * 9.0/5.0) + 32;
   else
    degs = "Not a valid Scale";
   </cfscript>

   <cfreturn degs>

  </cffunction>

</cfcomponent>
```

Listing 3

```
<!--- // set some local variables with values for conver-
sion --->
<cfset temp = 24>
<cfset scale = "C">

<!--- // invoke the web service for temperature conversion
--->
<cfinvoke method="doTempConvert"
    returnvariable="degs"
    webservice="http://localhost:8500/cfdj/com/tempcon-
vert.cfc?wsdl">
     <cfinvokeargument name="ATemp" value="#temp#">
     <cfinvokeargument name="ItsScale" value="#scale#">
</cfinvoke>

<!--- // output reponse from Web service --->
<cfoutput>
  Original temp: #temp# #scale# <br>
  Converted to: #degs# F <br>
</cfoutput>
```

Listing 4

```
<cfscript>
  // set some local variables with values for conversion
  temp = 24;
  scale = "C";

  // invoke the web service for temperature conversion
  ws = createObject("webservice",
"http://localhost:8500/cfdj/com/tempconvert.cfc?wsdl");
  degs = ws.doTempConvert(temp, scale);

  writeOutput("Original Temp:" & temp & " " & scale &
"<br>");
  writeOutput("Converted to: " & degs & " F");
</cfscript>
```

CODE LISTING

The code listing for this article is also located at

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

Using ColdFusion Components

Part 2

The new building blocks for all CF applications

BY BEN FORTA

Last month I introduced you to ColdFusion Components – CFCs for short. Following a brief introduction to the world of objects, we looked at CFCs and their syntax, and simple calling conventions using <CFINVOKE>. This month we'll continue this topic.

Using CFCs

In the last issue we created a CFC called *browser* (used to perform browser identification tests) and invoked methods in it like this:

```
<CFINVOKE COMPONENT="browser"
          METHOD="IsIE"
RETURNVARIABLE="result">
```

The IsIE method (which checks to see if Microsoft Internet Explorer is the browser in use) in the browser component takes an optional argument named browser and defaults to the CGI variable of the browser in use if not provided.

There are two different ways to pass arguments to component methods. The first is simply to append an attribute to the <CFINVOKE> call:

```
<CFINVOKE COMPONENT="browser"
          METHOD="IsIE"
RETURNVARIABLE="result"
BROWSER="#browserid#">
```

Alternatively, the <CFINVOKE ARGUMENT> tag may be used, as follows:

```
<CFINVOKE COMPONENT="browser"
          METHOD="IsIE"
RETURNVARIABLE="result">
   <CFINVOKEARGUMENT
NAME="browser"
VALUE="#browserid#">
</CFINVOKE>
```

The second format provides greater control over passing optional arguments (as <CFINVOKEARGUMENT> may be used conditionally or within a loop if needed). Both calling conventions do the exact same thing, so use whatever suits you best.

Alternate Invocation Options

<CFINVOKE> provides a simple way to invoke component methods, but it's by no means the only way to do so. <CFINVOKE>, as used above, actually does two things:

1. It loads the component.
2. It then invokes a specific method.

Multiple <CFINVOKE> calls to the same component (perhaps to different methods within the component) will each be loaded separately. As such, any processing performed within the component (initializing variables, for example) must be performed for each invocation individually, and processing is not shared between those instances.

Suppose you created a "user" component that you then invoked to obtain a user name and then invoked again to obtain a user e-mail address. Your calls might look something like this:

```
<CFINVOKE COMPONENT="user"
          METHOD="GetFullName"
USERID="#FORM.userid#"
RETURNVARIABLE="name">
<CFINVOKE COMPONENT="user"
          METHOD="GetEMail"
USERID="#FORM.userid#"
RETURNVARIABLE="email">
```

Notice that you'd need to pass the user ID to each invocation (as the second one would have no knowledge of the first). In addition, within the component the actual lookup would have to happen twice – once for each invocation.

A better solution would be to treat the CFC as an object, loading it using the <CFOBJECT> tag:

```
<CFOBJECT COMPONENT="user"
          NAME="objUser">
<CFINVOKE COMPONENT="#objUser#"
          METHOD="Init"
USERID="#FORM.userid#">
<CFINVOKE COMPONENT="#objUser#"
          METHOD="GetFullName"
RETURNVARIABLE="name">
<CFINVOKE COMPONENT="#objUser#"
          METHOD="GetEMail"
RETURNVARIABLE="email">
```

While <CFINVOKE> both loaded the component and executed a method, here those two steps have been separated from each other. <CFOBJECT> loads the component into an object (named using the NAME attribute). The <CFINVOKE> tags then use that existing object to invoke specific methods.

The first <CFINVOKE> call invokes an Init method, passing the user ID to the component. Now subsequent <CFINVOKE> calls need not pass the user ID, as the same instance of the component is being used in each call and it already knows the ID. Furthermore, within the component any user processing (perhaps reading the user from a database) would have to occur just once instead of for every method invocation.

The fact that CFCs may be used as objects creates another interesting opportunity. Although ColdFusion custom tags may not be used within <CFSCRIPT> blocks (because tags may not be used in <CFSCRIPT>), objects can be, because there is a function equivalent of <CFOBJECT> – the CreateObject() function. Since components can be loaded as objects, they can be used within <CFSCRIPT>. Here's the <CFSCRIPT> version of the previous code block:

```
<CFSCRIPT>
objUser=CreateObject("compo-
nent", "user");
objUser.Init(FORM.userid);
name=objUser.GetFullName();
email=objUser.GetEMail();
</CFSCRIPT>
```

As you can see, components can be used exactly like objects, and the <CFSCRIPT> syntax is particularly useful to developers who have worked with objects in other development languages.

There are other invocation options too:

- HTML forms may be submitted to CFC files directly (specifying the CFC file as the ACTION).
- URLs may invoke CFCs directly, passing the method and any parameters as URL parameters in the query string.
- Flash MX ActionScript code (using the NetServices functions) can invoke component methods (essentially a client/server application, Flash on the client invoking CFCs on the server).
- CFCs may be consumed as Web services (more on that later).

As you can see, CFCs are very flexible and very accessible. In fact, a single well-written CFC will likely be used in all sorts of ways – and often in ways you never anticipated.

Note: As a rule, within a CFC your code should never make any assumptions about the environment it's running in and where requests are coming from. Even though CFC code has access to all scopes, you're better off never assuming that any particular scope or variable exists or that requests are being made by specific clients or specific platforms. This way your code will be far more portable.

Data Abstraction

We've now started talking about data abstraction (something I first mentioned last month). Look at the preceding code – if the user ID is passed in one invocation, how do subsequent invocations have access to it?

To explain this, take a look at the user component (file user.cfc) in Listing 1. It's by no means complete, but it is working code, and will help explain how data abstraction works.

The first thing you'll notice is that there is code that is not in any function:

```
<!--- Initialize variables --->
<CFSET THIS.initialized=FALSE>
```

Any code inside a component but not inside a particular function is constructor code, code that's automatically executed when a component is first loaded. This line of code sets a flag called *initialized* to FALSE. The flag is stored in the THIS scope; a special scope within a component, it persists as long as the instance of the component persists, so anything stored within it is available to all invocations. Once initialized is placed in the THIS scope, that flag is available to subsequent method calls.

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
chief operating officer
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 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 / JDJ store
manager, customer relations / JDJ Store
Anthony D. Spitzer tony@sys-con.com
customer service representative
Margie Downs margie@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
chief financial officer
Bruce Kanner bruce@sys-con.com
assistant controller
Judith Calnan judith@sys-con.com
accounts receivable
Jan Braidech jan@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

The first method, Init, requires that a user ID be passed to it. It then retrieves the user record for that ID using a <CFQUERY>. If the record was retrieved (i.e., the ID was valid and usable), then Init saves a copy of it in the THIS scope and sets the initialized flag to TRUE.

The next two methods return data. GetFullName returns the user's full name (concatenating the first and last names) and GetEMail returns the e-mail address. Both methods first check to see if THIS.initialized is TRUE (if it isn't, Init was never called or the Init failed, and no user record would be present to return), and then return data from the query saved in the THIS scope. This way data needn't be reread each time it's needed.

The benefit of code like this is that, once initialized, user information is readily available throughout an application. There's no need to refer to underlying databases. If databases changed, or if additional business logic were needed or special formatting were to be applied, all that code could be localized inside the component, transparent to any invocations.

You'd need more code in the user component, of course. It would also need to know how to add, update, delete, search, and more. But you get the idea.

Persistent CFCs

It gets better. You now know that components can be loaded into objects (using the <CFOBJECT> tag or the CreateObject() function). Just as they can be loaded as standard variables (in the VARIABLES scope), they can also be loaded into persistent scopes (like SESSION or APPLICATION).

For example, if your application required users to log in, you might want to keep an instance of their user object in a SESSION variable so that it would persist for as long as the SESSION did. Look at this code:

```
<CFOBJECT COMPONENT="user"
          NAME="SESSION.objUser">
<CFINVOKE COMPONENT="#SESSION.objUser#"
          METHOD="init"
          USERID="#FORM.userid#">
```

Simple as that. Now any application code could invoke methods within the object in the SESSION scope:

```
<CFINVOKE COMPONENT="#SESSION.objUser#"
          METHOD="GetFullName"
          RETURNVARIABLE="name">
```

This will work for APPLICATION too if needed. The object will persist for as long as the scope does, and any data in the component's THIS scope will persist too.

Using Inheritance

Inheritance is a subject that really deserves a column unto itself, but here are the basics.

You've created a user object (file user.cfc) that abstracts all user processing. Now you need an administrator object that does everything the user object does plus a bit more. Rather than copy all the code into a new administrator.cfc file, you could create your component like this:

```
<CFCOMPONENT EXTENDS="user">
...
</CFCOMPONENT>
```

EXTENDS implements inheritance. As administrator.cfc extends user.cfc, everything in the user component will be accessible in the administrator component as well (even the THIS scope). Within the administrator component you could:

- Create additional methods that will be available only in administrator.
- Overwrite specific user methods by creating methods in administrator with the same names as methods in user.

Inheritance is an advanced concept, so I won't go into more detail just yet. But when you need it, know that it's there waiting for you.

Creating Web Services

Web services are all the rage now. Web services are simply the next generation of distributed computing, a mechanism by which to create and use parts of applications anywhere, even remotely. In Web service lingo, Web services are *produced* (made available for use) and then *consumed* (used) by other applications.

Web services aren't a ColdFusion invention. In fact, they're one of the few technologies that just about every major player in the space is backing and supporting. .NET uses them extensively, and J2EE applications support services too. This broad industry support means that Web services are going to become very important, they are going to succeed.

In addition, Web services employ several key technologies, all of which are open and standard:

- HTTP
- XML
- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description File)
- UDDI (Universal Description, Discovery, and Integration)

ColdFusion MX makes consuming Web services incredibly simple – all it takes to consume a service is a single CFML tag (the same <CFINVOKE> tag you've already seen).

I'll be covering Web services extensively in a future column, but for now I just want to explain how they're created in ColdFusion.

Are you ready? Hold tight – this gets really complicated. To create a Web service (that may be accessed by remote applications, not just ColdFusion), simply add the following to your <CFFUNCTION> tags:

```
ACCESS="remote"
```

That's it! By specifying that methods are accessible remotely, ColdFusion will make them accessible as Web services (even generating the required WSDL automatically on-the-fly).

More on this in future columns.

Securing Components

The last subject I need to mention is security. Custom tags don't need special security options. They're accessible by ColdFusion only locally so they're as secure as the rest of your ColdFusion code. But CFCs are accessible as Web services, which means they may be accessed remotely too. So how can you secure them against unauthorized access?

CFCs actually feature two security options that may be used individually or together:

1. Each method may take an optional ACCESS attribute (as just seen). By default, ACCESS="public", which means that the method is available to all ColdFusion code on your own server (just like custom tags). To make components accessible remotely, you must explicitly state ACCESS="remote". (Additional ACCESS levels control what ColdFusion code on your server may access methods, if needed.)
2. Methods may also have associated ROLES – groups of users that requesters must have been authenticated as in order to invoke a method. By default, no ROLES are used, but if needed this attribute provides a mechanism to grant or deny access based on login information.

So, by default, CFCs are as secure as custom tags. But if you need greater security (to manage greater accessibility), the support for that is there, ready for you to use.

Summary

As you've just seen, ColdFusion Components are incredibly powerful and flexible. They provide the power of objects while retaining the simplicity of CFML (supporting advanced object features such as inheritance and constructors), can be invoked and used in all sorts of ways, can be made to persist if needed, and can even be accessed as Web services. CFCs are the new building blocks that all ColdFusion applications should be built with. Learn them, use them, and take your code to a whole new level.

BEN@FORTA.COM

ABOUT THE AUTHOR
Ben Forta is Macromedia's senior product evangelist and the author of numerous books, including ColdFusion 5 Web Application Construction Kit *and its sequel,* Advanced ColdFusion 5 Development. *Ben is working on several new titles on ColdFusion MX. For more information visit www.forta.com.*

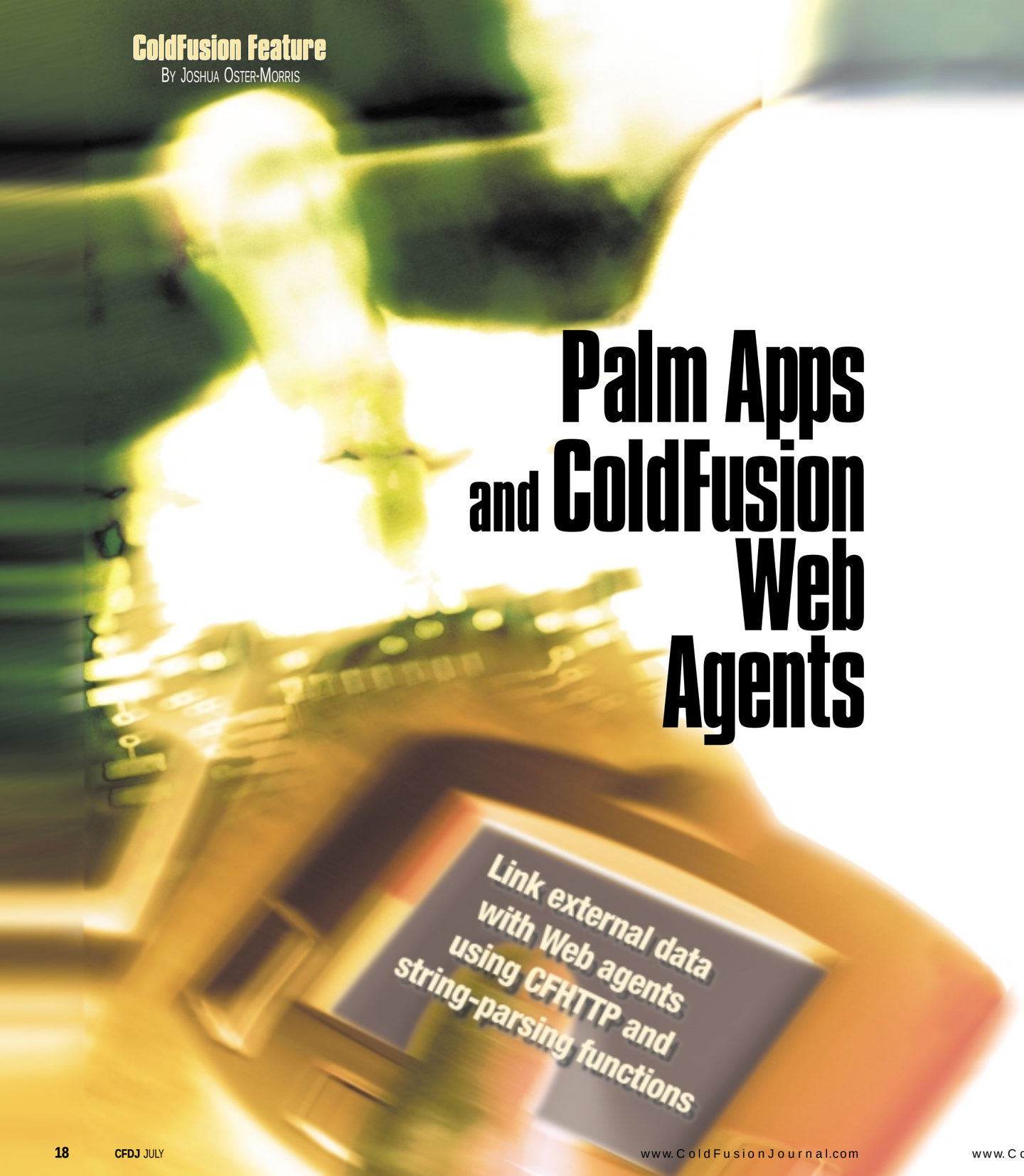
Listing 1

```
<CFCOMPONENT>

   <!--- Initialize variables --->
   <CFSET THIS.initialized=FALSE>

   <!--- Init user --->
   <CFFUNCTION NAME="init"
               RETURNTYPE="boolean">
      <CFARGUMENT NAME="userid"
                  TYPE="numeric"
                  REQUIRED="true">

      <!--- Get user --->
      <CFQUERY DATASOURCE="dsn"
               NAME="userRec">
       SELECT *
       FROM users
       WHERE userid=#userid#
      </CFQUERY>

      <!--- Check got a user --->
      <CFIF userRec.RecordCount IS 1>
         <!--- Save user record --->
         <CFSET THIS.userRec=userRec>
         <!--- Set init flag --->
         <CFSET THIS.initialized=TRUE>
      </CFIF>

      <CFRETURN THIS.initialized>
   </CFFUNCTION>


   <!--- Get user name --->
   <CFFUNCTION NAME="GetFullName"
               RETURNTYPE="string">

      <!--- Init return variable --->
      <CFSET result="">

      <!--- Check initialized --->
      <CFIF THIS.initialized>
         <CFSET result="#Trim(THIS.UserRec.NameFirst#)#
#Trim(THIS.UserRec.NameLast#)#">
      </CFIF>

      <CFRETURN result>
   </CFFUNCTION>


   <!--- Get user email --->
   <CFFUNCTION NAME="GetEMail"
               RETURNTYPE="string">

      <!--- Init return variable --->
      <CFSET result="">

      <!--- Check initialized --->
      <CFIF THIS.initialized>
         <CFSET result=Trim(THIS.UserRec.email)>
      </CFIF>

      <CFRETURN result>
   </CFFUNCTION>

</CFCOMPONENT>
```

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

By Joshua Oster-Morris

Palm Apps and ColdFusion Web Agents



We live and work in an increasingly portable world. As programmers, most of us remember the large steel boxes of the '80s and the even larger steel and glass fixtures of the '70s. We wrote software that our users accessed through those venerable VT100 terminals. Those terminals represented portability. These days, portable means the computer fits in your pocket. And if it can't get on the Internet without wires, then it's clunky.

This reality has yet to catch up to the reality that data throughput with these portable wireless devices doesn't allow us to browse the Web as though we're on the T1 at the office. The Palm solution to this problem is its Web clipping protocol, described wonderfully by Paul J. Elisii in the April 2000 issue of ***CFDJ*** ("ColdFusion in the Palm of Your Hand," Vol. 2, issue 4). Devices supporting Web clipping include the new Palm i705, the Palm VII, and several off-brand devices, including mobile phones and other PDAs.

As CF developers we can generate dynamic content for these devices as we would any other CF app. I've found this useful but limiting. I want to add data from outside my direct sphere of influence to my Palm apps, much as I can with a normal Web app by hyperlinking to an external source. You can't just hyperlink, because Web clipping apps require special instructions in order to render in the Palm OS. This means we need to massage the data. To do this we create Web agents that browse the Web for us on the server side. Next, we parse the returned content for the specific information we want to redirect to the Palm and format it for the Web clipping client device.

ColdFusion allows us to write Web agents with the CFHTTP tag. CFHTTP can make requests to a Web server using either the GET or POST method. With POST we can simulate a Web form submission, post a cookie, upload a file, or send a CGI variable directly to the server. This can be especially useful when trying to convince an outside Web server to return the data you want.

Figure 1 Palm application

To illustrate this technique, I'll walk you through the server side of an application I wrote to check movie times at local theaters and read reviews and plot abstracts (see Figure 1).

Let's examine the initial CFHTTP command.

```
<cfhttp url="***URL HERE***" method="get"
></cfhttp>
```

This command retrieves the movie times from a source on the Internet – in this case the local newspaper's Web portal (see Figure 2) – and stores it in the variable CFHTTP.FileContent. The CF server selects this variable automatically. See Listing 1 for an example of the code retrieved and Listing 2 for what it will look like postprocessing.

At this point I need to begin the process of removing all unwanted code from the document just downloaded. This process is made easier by the fact that HTML is a tag-based language designed to be parsed by the client. The </> format of HTML tags conveniently identifies them for simple modification or removal. Listing 3 shows how I start cutting out parts of the file I don't want.

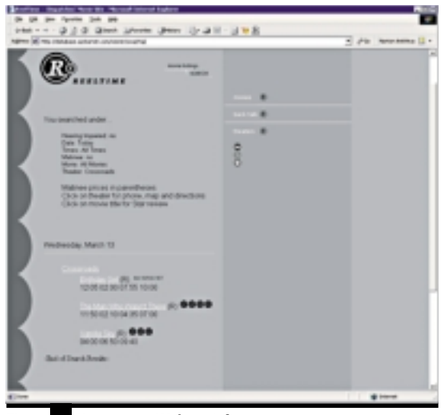


Figure 2 Newspaper Web portal

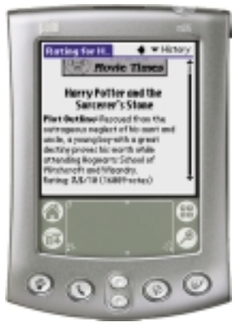


Figure 3 Plot abstract and rating

Figure 4 Final output

The snippet below uses three of ColdFusion's string-parsing functions in conjunction:

```
<cfset results =
right(CFHTTP.FileContent,
  len(CFHTTP.FileContent) –
  (find('</UL>',CFHTTP.FileContent,1) +
4))>
```

The RIGHT function truncates a string, cropping everything to the right of a point in the string. It has two arguments. First is the variable that's being truncated, in this case the returned Web page. Second is the position, as an integer, that becomes the new beginning of the string. This example shows how I used the LEN function to get the total length of the variable I'm truncating and then use the FIND function to find a certain point in the document. LEN returns an integer. FIND also returns an integer, but it has three arguments:

1. The string being searched for
2. The string being searched
3. The starting point of the search

In this example I add 4 to the outcome of the FIND function to move the pointer to the end of the string I'm searching for. Finally, I subtract that integer from the total length of the page returned.

The two snippets that follow show how I go about cutting out all the hyperlinks in the returned page – by looping through the page until they're all gone. This is important, because the hyperlinks will point to pages that the Palm won't be able to render.

```
<cfloop condition="find('<a
href',results) GT 0">
  <cfset a = find('<a href',results) -1>
  <cfset b = find('>',results,a) >
  <cfset results = left(results, a) &
  right(results,len(results)-b)>
</cfloop>

<cfloop condition="find('</a>',results)
GT 0">
  <cfset a = find('</a>',results) -1 >
  <cfset b = a + 4>
  <cfset results = left(results, a) &
  right(results,len(results)-b)>
</cfloop>
```

The following code shows how I go about cutting out all the <img> tags in the returned page – by looping through the page until they're all gone. This is important because the images will use up the very precious bandwidth of the already encumbered Web clipping connection.

```
<cfloop condition="find('<img',results)
GT 0">
  <cfset a = find('<img',results) -1>
  <cfset b = find('>',results,a) >
  <cfset results = left(results, a) &

right(results,len(results)-b)>
</cfloop>
```

The first code snippet below stores the details of each movie in a separate variable, while the second adds a hyperlink, a separate page on my server that goes to IMDB.com and retrieves a plot abstract for the movie and its rating (see Figure 3). Listing 4 then sends the final output to the Palm device (see Figure 4).

```
<cfset numofmovies = 0>
<cfloop condition="find('<p>',bot) GT 0">
  <cfset numofmovies =
  incrementvalue(numofmovies)>
  <cfset a = evaluate(find('<p>',bot)
  +3)>
  <cfset b = evaluate(len(bot) + a) >
  <cfset setvariable('mov' & numofmovies,
         left(bot, a))>
  <cfset bot=mid(bot,a,b)>
</cfloop>

<cfloop from="1"
        to="#evaluate(numofmovies-1)#"
        index="ftar">
  <cfset rate = left(evaluate('mov' &
  ftar),
         find(' (',evaluate('mov' &
  ftar))-1) >
  <cfset setvariable('mov' & ftar,
         replace(evaluate('mov' & ftar),
         rate,
         '<a href="rate.htm?movie=' &
         urlencodedformat(trim(rate)) &
'">' &
         rate & '</a>'))>
</cfloop>
```

• • •

Though certainly more complex than just adding a hyperlink, it's the cost of doing business. With a little experience, string parsing becomes an easy task. The art will slowly fade as more data sources offer their data in XML format for easy consumption, but until then we'll just have to parse away.

About the Author

Joshua Oster-Morris has been writing software since age 6, when he began his first computer game on his TRS-80. Since then he's deployed many mission-critical database applications using dBase, FoxPro, C++, and, most recently, ColdFusion and Java.

@ JOSH@CRAFTYCODER.COM

Listing 1

```
<HTML>
<HEAD>
<TITLE>ReelTime -- Dispatches' Movie Site</TITLE>
</HEAD>
<body background="/reeltime/bgi.gif" marginheight=0
 marginwidth=0 topmargin=0 leftmargin=0
 vlink="#ffffff" link="#ffffff">
*****many lines of HTML trimmed off*****
******First tag that is used for trimming*****
</UL>
<BR>
<BR>
<BR>
<p>Wednesday, April     24<hr>
<ul>
<a href="***URL***">Century Gateway 12</a><ul>
<a href="***URL***">A Walk to Remember</a> (PG)
<img src="***URL***"><br>
(02:55)
(05:10)
(07:20)
(09:25)
<p>
<a href="***URL***">Big Fat Liar</a> (PG)
<img src="***URL***"><br>
(01:30)
(03:30)
(05:30)
(07:30)
(09:20)
<p>
<a href="***URL***">Black Hawk Down</a> (R)
<img src="***URL***"><br>
(01:50)
(05:05)
(08:00)
<p>
</ul>
</ul>
</font>
<p>
-End of Search Results-
*****many lines of HTML trimmed off*****

</body>
</html>
```

Listing 2

```
<HTML>
<HEAD>
<meta name="PalmComputingPlatform" content="true">
<title>Movie Times</title>
</HEAD>
<body>

<center><img src="file:MOVIES.pqa/top.gif"></center>
03:17 PM <BR>
<p>Wednesday, April     24<hr>
<b>
Century Gateway 12</b><br><br> <a
href="rate.htm?movie=A%20Walk%20to%20Remember">
A Walk to Remember</a> (PG) <br>
(02:55)
(05:10)
(07:20)
(09:25)
<p>
<a href="rate.htm?movie=Big%20Fat%20Liar">
Big Fat Liar</a> (PG) <br>
(01:30)
(03:30)
(05:30)
(07:30)
(09:20)
<p>
<a href="rate.htm?movie=Black%20Hawk%20Down">
Black Hawk Down</a> (R) <br>
(01:50)
(05:05)
(08:00)
<p>
<a href="rate.htm?movie=Collateral%20Damage">
Collateral Damage</a> (R) <br>
(01:45)
(04:25)
(07:05)
(09:30)
<p>
</font>
<p>
-End of Search Results-
</body>
</html>
```

Listing 3

```
<cfset results=right(CFHTTP.FileContent,
  len(CFHTTP.FileContent) –
  (find('</UL>',CFHTTP.FileContent,1) + 4))>
<!-- trim left of </UL> -->
<cfset results=right(results,
  len(results) - (find('<BR>',results,1) + 3))>
<!-- trim left of first <BR> -->
<cfset results=right(results,
  len(results)-(find('<BR>',results,1) + 3))>
<!-- trim left of second <BR> -->
<cfset results=left(results,
  len(results)-(len(results)-
  (find('-End of Search Results-',results)+22)))>
<!-- trim right of first -End of Search Results- -->
<cfset results= replace(results,'<ul>','<b>','one')>
<!—replace first <ul> with <b> -->
<cfset results= replace(results,
  '<ul>','</b><br><br>','one')>
<!—replace next <ul> with </b><br><br> -->
<cfset results= replace(results,'<ul>','','all')>
<cfset results= replace(results,'</ul>','','all')>
<!—remove all other <ul>s -->
```

Listing 4

```
<HTML>
<HEAD>
<meta name="PalmComputingPlatform" content="true">
<title>Movie Times</title>
</HEAD>
<body>
<div align="center">
  <img src="file:MOVIES.pqa/top.gif">
</div>
<cfoutput>
#timeformat(now(),'hh:mm tt')# #trim(top)#
<cfloop from="1" to="#numofmovies#" index="ftar">
  #left(trim(evaluate('mov' & ftar)),
  len(trim(evaluate('mov' & ftar)))-3)#
  <cfif isdefined('rat' & ftar)>
    <cfif trim(evaluate('rat' & ftar)) is not ''>
      <br>Rating: #trim(evaluate('rat' & ftar))#
    </cfif>
  </cfif>
<p>
</cfloop>
#trim(bot)#
</cfoutput>
<br>
</body>
</html>
```

CODE LISTING

The code listing for this article is also located at

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

What Really Leads to **Successful Software?**

It's the architecture, of course!

BY HAL HELMS

ColdFusion MX is a complete, from-the-ground-up rewrite of an existing piece of software, shifting its architectural base from a purely interpreted language sitting atop C, its base language, to one that compiles into Java code. That's quite a shift.

For months Macromedia has been reassuring developers that, whatever may happen beneath the covers, nothing changes about our code. In one sense that's absolutely true. Take code written under a previous version of ColdFusion and – almost without fail – it will run perfectly with ColdFusion MX. In that sense "nothing changes" is an accurate reflection of reality.

On another level "everything changes" is an equally valid description of the new world that CFMX has led us into. ColdFusion's new foundation of Java offers us new capabilities, such as using JSP custom tags in ColdFusion pages. Charlie Arehart (www.systemanage.com) has been very helpful in pointing out in his articles and seminars new ways of utilizing the CF–Java connection.

Certainly ColdFusion's native XML parsing will open up new opportunities for us, and the use of CFCs promises to make coding simpler, easier, and more robust – a nice trifecta.

I've talked with many developers who sense that everything does indeed change with ColdFusion MX, but can't fully articulate what those changes might be. Often this conviction is accompanied by a certain dread, a fear that the good old days of coding in ColdFusion may be over, that the door of opportunity that ColdFusion represented is rapidly closing, to be replaced by a guarded and armored door with a sign reading, "Dare not enter save ye have a propeller on thy head." Or words to that effect.

This would be a tragedy, as the bad old days of the propeller heads in charge are little missed. (See Alan Cooper's wonderful *The Inmates Are Running the Asylum* for a brilliant perspective from a truly qualified propeller head.) Still, I do think that ColdFusion MX will bring in changes that we don't fully apprehend.

We've all heard the probably apocryphal curse, "May you live in interesting times," and the post-MX world may prove very interesting. But we may take heart in another old saw that tells us that every danger is also an opportunity. That's what I'd like us to consider in this article.

In my training classes (www.halhelms.com) I use a wonderful quote by MIT professor Hal Abelson: "If we can dispel the delusion that learning about computers should be an activity of fiddling with array indexes and worrying whether x is an integer or a real number, we can begin to focus on programming as a source of ideas."

Many people, too long kept at bay by the old-style propeller-head guards, will readily agree with the first part of this statement. They love the fact that ColdFusion doesn't obsess over variable types and appreciate that sanity is restored by, for example, beginning to count arrays at one instead of zero. But Abelson doesn't stop with a mere indictment of excessive fussiness; he adjures us to view programming as a source of ideas – and there's the rub, to borrow a phrase from another author.

After we get over the delusion that programming is about twiddling bits, we're still left with the much harder matter of ideas – of thinking clearly – and that's no small matter.

I recently spent several hours with a company that wanted me to consult with them on building a large Fusebox application. This was an introductory meeting in which they were to sketch the outline of the project and fill in the details on what had been done to date. My job was to absorb information and lay out a plan for going forward.

After initial introductions, they referred me to two giant whiteboards that looked truly impressive. They were covered with words and symbols, and atop each board, in bold letters, were the words DO NOT ERASE!!! They began to ask me questions about nesting circuits, how to reuse fuseactions, and how nested layouts should be handled using Fusebox.

Now these are excellent questions, but something was lacking. Our discussion was highly abstract and theoretical, necessitating virtually every statement to be modified with the unhelpful phrase, "Depending on the application…."

I became concerned that there was no concrete thing we could talk about. Even though I spend a good deal of time theorizing, I'm very cognizant that what matters are real applications with real deadlines and real budgets. If theory doesn't help developers successfully deploy projects on time and on budget (or preferably below both), it's no more than mental gymnastics.

"Have you gone through a prototype, and if so can I see it?" I asked.

The room grew rather quiet. They hadn't gone through a prototype, or rather, they had a few pages mocked up and felt that was going far enough. "We don't have a full prototype, if that's what you mean," I was told.

"How do you know what you're building?" I asked. I was told that the whiteboards represented the "architecture" of the application. I saw various menus and a few logical groupings but nothing I'd call architecture.

"We know it's not complete," they said. "That's why we want your help – to design a good architecture."

I tried an illustration: "With a physical building, the choice of an architecture depends on a number of factors, including items highly important to clients (shape, size, colors, etc.) and those underlying issues that a good architect is responsible for understanding, such as structural engineering, electrical, plumbing, and so on. Until the architect knows what the building will look like, issues such as the stress on load-bearing beams simply can't be determined. It's really the same with a software 'building.' That's why I really urge you to begin with a prototype: it will reveal issues and questions that aren't obvious in the abstract – issues, in fact, that will make or break your project."

It was no good; I wasn't convincing them at all. The more we talked, the more it became clear to me that what they wanted from me was an all-purpose architecture that would fit a wide variety of projects and would be adaptable to the changes that are inevitable in a software project. I finally had to say that I was very sorry, but wouldn't be able to help them with the project. I wished them the best and we parted.

My purpose in relating this is not to denigrate these folks. I have every reason to believe that they are hard-working, conscientious, and passionate about creating successful software. But they lacked knowledge of what leads to successful software, and assumed, in the absence of knowledge, that they could hire someone to produce an all-purpose architecture. And that, neither I, nor anyone I know, can do.

If this were an aberration I wouldn't mention it, but I've become convinced that it's all too often the rule rather than the exception. We developers want to "get coding." Or at least get the database design out of the way. Do something real. Something concrete. We aren't treating programming "as a source of ideas," but are back to throwing code against the wall to see what sticks.

The irony is that all this "concrete" work has no firm basis, no real architectural foundation. And without that foundation the structure being built will almost certainly fall.

I seem to have gone far afield from my initial sentences about the conjunction of Java and ColdFusion, but I see a strong connection between my first thoughts and this tangent about a client. I agree that ColdFusion's close relations with Java will change a great deal about how we do things, and I want to see that change benefiting us developers and our clients.

For that to happen we must examine the idea of software architecture; we must make sure that we think first and program after our thoughts are clear. This is neither easy nor fun, at least not initially. Developing a software architecture sometimes to involve nothing so much as finding ourselves along conceptual dead-ends that seemed promising when the path first appeared.

When we speak of "thinking clearly," what we mean is the courage and fortitude to toss out our ideas, to "go back to the drawing board" – literally in our case. Throwing out hours or days of plans isn't enjoyable, no matter how positive our outlook. It feels like a loss, and indeed it is. We persevere because we've found that undergoing some pain initially gives us a much, much greater chance for success later on.

• • •

Next month I'll write about a software architecture called Model View Controller, or MVC. MVC was initially created by the bright lights at Xerox Palo Alto Research Center (PARC), and has been used for many complex, sophisticated applications. An adapted version of it forms the recommended Model 2 of the J2EE architecture. In my classes I teach MVC as well suited to certain types of applications and show how it can be used with Fusebox. I have to warn you that parts of MVC aren't intuitive – again, at least not initially. The advantages of mastering this architectural model, though, are significant and can help ensure that our projects don't fail for lack of a robust foundation on which to stand. So eat your Wheaties, and I'll see you next month!

ABOUT THE AUTHOR

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

@ HAL@FUSEBOX.ORG

A Quick and Easy Web Service with ColdFusion MX

Using the new features

BY KEVIN SCHMIDT

ColdFusion MX is here, and with it come bundles of exciting new technology. Two pieces of this technology involve XML and Web services.

By constructing a simple Web service to syndicate new articles, you'll get a firsthand look at how ColdFusion MX uses these technologies.

Storing Your Content

In order to syndicate content, you first need a place to store it. This is where the database comes into play. For this example I used Microsoft's SQL Server, but you could just as easily use Access. I designed a very basic table called *tblNewsArticles* to hold the news article information. It has fields for an ID, whether or not the article is active, and finally, the actual content of the article (see Figure 1). Now that you have the database, you need to be able to add content to it.

Adding Content

Content is added to the database through a standard HTML form, and the simple one I've constructed for this example appears in Figure 2. The really neat and interesting stuff doesn't take place until you submit the form. Once it's submitted you must put all the information into an XML string. This can be done using the following code:

```
<cfset xmlstring =
"<article><title>#form.arti-
cletitle#</title><body>#form.art
iclebody#</body><date>#now()#"</
date></article>">
```

With the data in XML format, you're ready to insert it into the database. The following code takes care of the insert:

```
<cfquery name="insertxml" data-
source="yourdsn">
 INSERT INTO tblNewsArticle (
    vchrNewsArticle
    )

   VALUES (
    '#xmlstring#'
    )
</cfquery>
```

Now that you have the data in the database, you need to create the Web service to access and syndicate the content.

Creating Your Web Service

Your data is in the database and ready to go – how do you make it available as a Web service? CFMX introduces a new feature called *components*, which allows you to encapsulate functionality and make it available to many different clients, including Flash and Web services. It allows you to specify multiple methods and pass in arguments. For the purpose of our simple Web service, we'll use a component that has one method and doesn't require arguments.

You must use a component to create your Web service, and using the <cfcomponent> tags makes building Web services quick and easy. Use the <cfcomponent> tag to start the component; it will enclose the rest of the tags necessary to create the Web service.

```
<cfcomponent>
  <!--- content of tag goes
here --->

 </cfcomponent>
```

As you can see, using this tag is no different from using any other CF tag. The next step is to define the method of the component. This is done by using another new tag in CFMX, <cffunction>. The following code creates a method called *getnews* for the component:

```
<cfcomponent>
 <!--- content of tag goes
   here --->
 <cffunction name="getnews">

 </cfunction>
</cfcomponent>
```

You've designed the function; now you need to give it power to perform a task, in this case, retrieve the records from the tblNewsArticle table. The following code illustrates this:

```
<cfcomponent>
 <!--- content of tag goes
   here --->
 <cffunction name="getnews">
  <cfquery name="getnews" data
   source="yourdsn">
  SELECT something
  FROM tblNewsArticle
  WHERE bitActive = 1
  </cfquery>
 </cfunction>
</cfcomponent>
```

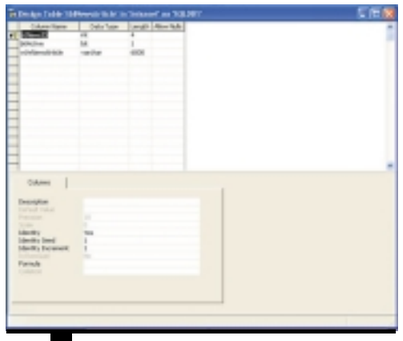


Figure 1 Example database

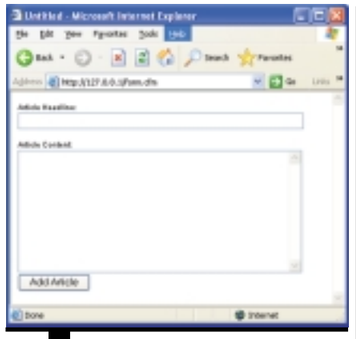


Figure 2 Standard HTML form

Your component will retrieve a recordset containing all the active news articles. It also needs to be able to return that recordset to the page or site that called it. Using the tag <cfreturn>, you specify the string or variable you want returned to the calling page or site; in this case you're going to return the getnews recordset:

```
<cfcomponent>
 <!--- content of tag goes
   here --->
 <cffunction name="getnews">
  <cfquery name="getnews" data
   source="yourdsn">
  SELECT vchrNewsArticle
  FROM tblNewsArticle
  WHERE bitActive = 1
  </cfquery>

  <cfreturn getnews>
 </cfunction>
</cfcomponent>
```

You're only one step away from turning this component into a Web service. You must add the access and returntype attributes to the function you created, and specify that the access is remote and the returntype is any, as follows.

```
<cfcomponent>
 <!--- content of tag goes
   here --->
 <cffunction name="getnews"
  access="remote" return
  type="any">
  <cfquery name="getnews" data
   source="yourdsn">
  SELECT
  FROM tblNewsArticle
  WHERE bitActive = 1
  </cfquery>

  <cfreturn getnews>
 </cfunction>
</cfcomponent>
```

Because you specify that the access is remote, CFMX automatically generates the WSDL (Web Services Description Language) file needed for the method. When you're finished, save your file with a new extension, .cfc. You're all set. You've created a component that can be called as a Web service

Accessing Your Web Service

You now need a way to access your Web service and display the results. Use the <cfinvoke> tag, also new to CFMX, to access the Web service. This tag is used to access local components as well as Web services. The first attribute you need to use with it is the webservice attribute, with which you'll specify the location of the Web service. The following code shows an example:

```
<cfinvoke
   webservice="http://127.0.0.1/
   getnews.cfc?wsdl">
```

We'll assume the component you created was saved as getnews.cfc. This component is made up of one method, getnews. In your <cfinvoke> tag you need to specify which method you want to call – in this case there is only the one. The following code can be used:

```
<cfinvoke webservice=http://
   127.0.0.1/getnews.cfc?wsdl"
   method="getnews">
```

When it's called, the getnews.cfc will return a recordset of all the active news items. To access the returned recordset, you need to specify a return variable in your <cfinvoke> tag. The returnvariable attribute is used with the name of the variable you want the information stored in. For example:

```
<cfinvoke webservice=http://
   127.0.0.1/getnews.cfc?wsdl"
   method="getnews" returnvari
   able= "news">
```

You now have a variable, news, that contains a recordset of all the articles in XML format, and you need to display them on your page.

Displaying the News Articles

When you're looping over the returned news recordset, you must first convert the XML to a usable format in order to display the articles. CFMX provides a host of tools to deal with XML, including the XmlParse function, which converts an XML string into an XML object made up of CFML structures and arrays. The following code could be used to convert the string:

```
<cfoutput query="news">
<cfset currentarticle =
 XmlParse(vchrNewsArticle)>
```

Once this is done, you'll have the current article in an XML object and be able to use the traditional dot notation to access the values inside the object. To display the current news article, you could use the following code.

```
<cfoutput query="news">
<cfset currentarticle =
 XmlParse(vchrNewsArticle)>

#currentarticle.article.name.xml
text#<br>

#currentarticle.article.body.xml
text#<br>

#dateformat(currentarticle.arti-
cle.date.xmltext,
"mm/dd/yy")#<br>
 <br>
 </cfoutput>
```

For each record in the query, the vchrNewsArticle field will be loaded into an XML object and each piece of the news article will be output.

Moving On

This brief example showing how to use Web services with the new features in CFMX barely scratches the surface of what you can do with components. As CFMX nears its full release, components will take center stage as one of the best and brightest new features.

ABOUT THE AUTHOR

Kevin Schmidt is a Macromedia-certified ColdFusion developer and author of Macromedia ColdFusion 5: Training from the Source. *He is manager of the Des Moines ColdFusion User Group.*

@SCHMIDT@HUNGRYCOW.COM

By Jeff Peters

Using Querysims to Analyze Log Files

Query simulations, or querysims, are a means of simulating returned records from a database when no database exists. This article explores a method of using the <cf_querysim> tag to create an easy approach to custom log file processing.

Querysims 101

The <cf_querysim> custom tag was developed by Hal Helms as a tool to make development of Fusebox applications less linear. The idea was to disconnect the front-end CFML development from the back-end database and query development. To do this, <cf_querysim> provides a way to generate ColdFusion recordsets without querying a database. Instead, lines of text data are converted into a recordset.

As an example, let's imagine we're building a site that needs to display a list of employees. We need to retrieve each employee's first and last names, employee identification number, department, and supervisor ID number from the database. A <cfquery> to satisfy this requirement is shown in Listing 1. All the listings in this article have Fusedoc blocks at the top to document the function of the template. More information on Fusedoc can be found at www.fusebox.org or www.halhelms.com.

An extraordinarily useful custom tag

Listing 2 details dspEmployees.cfm, a template that produces a table based on the data returned by qryGetEmployees.cfm.

To tie the two together, we include them in a calling template, exampleOne.cfm, shown in Listing 3. This technique separates the back-end data portion of the code from the front-end display portion, another idea used extensively in Fusebox.

The end result, produced by running exampleOne.cfm, is shown in Figure 1. This is familiar territory for most ColdFusion developers. The twist comes when we want to develop and test the display component of this example before the database exists. This allows us to continue development regardless of whether there's a database yet.

To accomplish this goal, we need a way to make qryGetEmployees.cfm produce output just as though the database was done. This is where the querysim custom tag comes in. Listing 4 shows a version of qryGetEmployees.cfm that creates a querysim of desired data. The first line inside the <cf_querysim> tag defines the name of the recordset that will be produced, the second line specifies field names, and the remaining lines specify the data.

When we run exampleOne.cfm using the new querysim, the output looks exactly the same as it did in Figure 1. The querysim has taken away the need for the database.

Common Uses for Querysims

As shown in the previous examples, querysims were developed to allow developers to get on with the work of creating an application's front end without having a complete database on hand. This means that the project's participants can work in parallel, reducing the calendar time required to build the application. ColdFusion coders can work on their side of the application, supported by querysims to represent live data, while database developers work independently on the back end. As query files are written, using SQL, they're put in the application in place of the querysims that stand in their stead.

Querysims can be useful in other ways as well. For example, most of us have had to build a form to be used to add and edit data. When adding a record, we need a blank form. When editing a record, we need the form to populate with data from the database. One typical solution to this problem is to create conditional logic for each input on the form, populating the input with data if a record is available, otherwise leaving the input without a value.

Querysims make this task much more manageable. We start with the idea that a form is always in edit mode. The only difference between creating a new record and editing an existing record is that, in the case

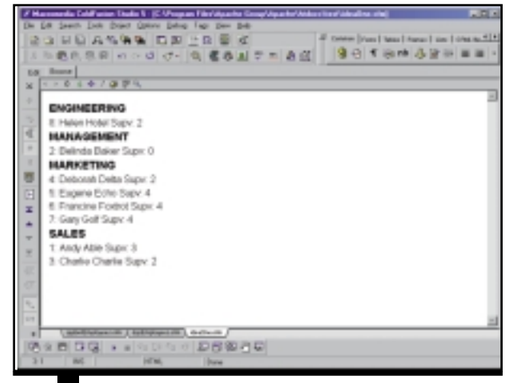


Figure 1 Employee output

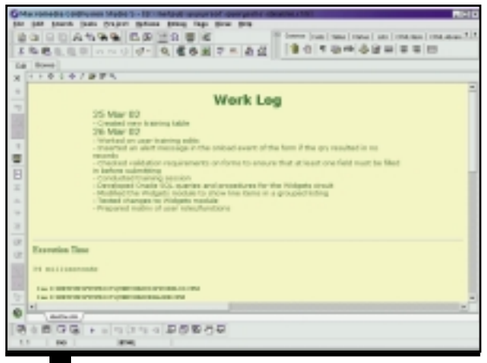


Figure 2 Work Log display (ideaOne.cfm)

of creating, we're really editing a record with all blank fields. So we create a single piece of conditional logic at the top of the form that checks to see if we're editing a record. If not, we create a recordset using <cf_querysim>. This recordset has one record with all blank fields. This way, the code that displays the record's values for editing won't throw an error for a creation action – the recordset always exists, regardless of whether we're editing an existing record or creating a new one. Listing 5 shows a simple example of this technique.

Notice that there is no conditional logic inside the form in Listing 5. All the work is done by the querysim. Regardless of whether we're creating or editing a user, we always deal with a recordset, so there's no need for cluttered conditional logic.

Parallel development and form manipulation are powerful uses of querysims, but something came up that led me to explore more ways to take advantage of them.

The Problem

Now that we've had a quick tour of querysims, I'll get into the subject problem for this article. I recently had a request to create a project status page for one of my clients. The request was to provide daily status reports on the project using a Web page.

The restrictions on creating such a page were interesting, though. The client asked that it be quick, easy, cheap, and attractive. Quick means "Don't spend much of my money putting it together," easy means "Don't spend much of my money by making it time-consuming to update," cheap means "Don't spend much of my money," and attractive means "You're not allowed to shove a plain text page at me."

For a bunch of developers, this should be an easy request. After all, everyone on my team can write HTML, so it would be an easy matter to pop up a page of HTML and let everyone edit it daily to add their progress notes. We certainly could have gone this way, but this particular client has a habit of changing his mind, particularly where layout-related things are concerned. So I fully expected him to change his mind at some point about how he wanted these daily updates presented. That, combined with my ingrained Fusebox thinking that tells me to separate data from process and presentation, led me to consider something different.

The Solution: Idea One

The approach was simply to create a query file with a querysim in it to contain the daily update log. The querysim would present the log data for a display file to render for the user. With this approach, if the presentation requirements changed, we could just change the display file. In addition, we'd be able to use the same query file as input to a variety of displays, just in case things got interesting.

The query file I worked up is shown in Listing 6. I refer to this as "Idea One" as it became the foundation for more ideas in the same vein.

Listing 7 shows the display file I used to process the log, and Figure 2 shows the log displayed in a browser, again using a calling file (ideaOne.cfm) to pull together the query and display files.

Left at this point, the solution might have been fine. However, the ways of Fusebox, once learned, aren't easily ignored. Having developers editing the log data right in the querysim definition made me a little nervous. Everyone on the project knew better than to mess around with the CFML and to simply edit the data inside the <cf_querysim> tag, but on the off chance that someone would slip a finger and accidentally delete the starting bracket on the </cf_querysim> closing tag, I decided I needed to keep the data somewhere other than embedded directly in the <cf_querysim> tag. Enter Idea Two.

The Solution: Idea Two

Probably the simplest part of the solution, Idea Two represents the true power of this approach. The idea is simple: separate the data from the <cf_querysim> tag through the use of <cfinclude>. Using this idea, the qryWorkLog.cfm file became two files. The first is qryWorkLog2.cfm, which is just qryWorkLog.cfm with a small modification to remove the data and replace it with a <cfinclude> tag. The second is WorkLog.txt, which contains the data removed from qryWorkLog.cfm. These two files are shown in Listings 8 and 9.

The end result is the same output as shown in Figure 2. Nothing has really changed about the data or how it's presented. On the back end, though, we now have a standalone text file that can be edited without fear of breaking the querysim code.

Having implemented this solution, I looked at WorkLog.txt and realized it was nothing more than a simple log file, much like those generated by Web servers. That realization led me back to some discussions from various listservs and newsgroups about Web statistics packages and parsing server logs. It occurred to me that the use of querysims represented an easy way to import a server log into a CF recordset for further processing. And so we go on to The Next Idea.

The Next Idea: Server Logs to Recordsets

The records in a querysim data file are pipe-delimited. That is, each field is separated from the next by a vertical pipe (or bar) character. Most server logs simply have spaces between fields, making them problematic to parse efficiently. In order to use the querysim tag, I'd have to take one of two approaches. I could either modify the querysim tag to parse the server log, or I could modify the server log to comply with the querysim tag's requirements. Because spaces aren't particularly good delimiters to begin with, I decided on the latter approach.

Fortunately, I do most of my work on servers that run Apache, so modifying the server log was really very simple. I went into the Apache configuration file, httpd.conf, and added the following line along with the other LogFormat lines:

```
LogFormat "%h|%l|%u|%t|\"%r\"|%>s|%b" pipedcommon
```

This defines a new log format called "pipedcommon", which is identical to the common server log format except that it uses pipes instead of spaces between fields. I then modified the CustomLog directive to use this new log format:

```
CustomLog logs/access.log pipedcommon
```

A quick restart of Apache and it was ready to go. Every request to the server causes a line to be written to the access log, so I made a few page requests to add lines to a new log file, creating the file in Listing 10.

Then I took a copy of the log file over to my ColdFusion test directory, where I had a new file waiting for it. This file, qryWebLog.cfm, is shown in Listing 11. It's identical in concept to the qryWorkLog2.cfm file seen in Listing 8, but the querysim has a different name and the field headings are altered to match the format of the server's access log. In addition, I've added a <cfdump> tag to the end of the file to quickly show that the server log has indeed been processed into a recordset.

For live use I created a display file (dspWebLog.cfm) and a calling file (nextIdea.cfm), similar to the examples shown earlier, to create an attractive display of the access log's data. These files are shown in Listings 12 and 13, and the output from running nextIdea.cfm is shown in Figure 3.

As you can see, the Web log is neatly displayed in the browser window, in just the format I specified. This is the launching point for whatever sort of log analysis you might wish to perform. Particularly with ColdFusion's query-of-query capability, you could do just about any sort of analysis you might want on this recordset.

Other Applications for Querysims

As you think about querysims, of course, more and more uses for them become apparent. You can take advantage of

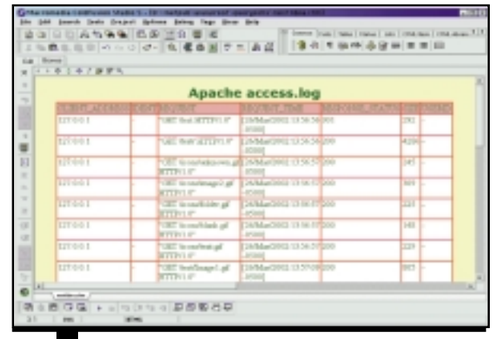


Figure 3 Web log display

<cf_querysim> any time you might want to convert text data into a recordset without worrying about a custom parser.

For example, you might want to create a bulk loader for text data. With <cf_querysim> loading the data into a recordset for you, loading the data into a database becomes a simple matter of looping over the recordset with a <cfquery> to insert the data. No doubt your imagination will be able to come up with its own uses for this extraordinarily useful custom tag

About the Author

Jeff Peters, a member of the Fusebox Council, is a consultant for Operational Technologies Services, Inc., Vienna, VA. He has been tinkering with things computerish for almost three decades now (www.GrokFusebox.com), and is the author (with Nat Papovich) of Fusebox: Developing ColdFusion Applications *(New Riders).*

@ JEFF@GROKFUSEBOX.COM

Listing 1: qryGetEmployees.cfm

```
<!---
<fusedoc fuse="qryGetEmployees.cfm" language="ColdFusion"
version="2.0">
  <responsibilities>
    I return a recordset of employee data.
  </responsibilities>
  <io>
    <out>
      <recordset name="qryGetEmployees">
        <number name="employeeID" />
        <string name="firstName" />
        <string name="lastName" />
        <string name="department" />
        <number name="supervisorID" />
      </recordset>
    </out>
  </io>
</fusedoc>
--->
<cfquery name="qryGetEmployees" datasource="personnel"
dbtype="ODBC">
  SELECT employeeID,
         firstName,
         lastName,
         department,
         supervisorID
    FROM Employees
ORDER BY department,
         employeeID
</cfquery>
```

Listing 2: dspEmployees.cfm

```
<!---
<fusedoc fuse="dspEmployees.cfm" language="ColdFusion" ver-
sion="2.0">
  <responsibilities>
    I display a table with employee information sorted by
department. For each employee, I display the full name and
the supervisor's ID.
  </responsibilities>
  <io>
    <in>
      <recordset name="qryGetEmployees">
        <number name="employeeID" />
        <string name="firstName" />
        <string name="lastName" />
        <string name="department" />
        <number name="supervisorID" />
      </recordset>
    </in>
  </io>
</fusedoc>
--->
<table>
<cfoutput query="qryGetEmployees" group="department">
  <tr>
    <td>
      <h3>#UCase(department)#
    </td>
  </tr>
  <cfoutput>
  <tr>
    <td>#employeeID#: #firstName# #lastName# Supv:
#supervisorID#</td>
  </tr>
  </cfoutput>
</cfoutput>
</table>
```

Listing 3: exampleOne.cfm

```
<cfinclude template="qryGetEmployees.cfm">
<cfinclude template="dspEmployees.cfm">
```

Listing 4: qryGetEmployees.cfm using <cf_querysim>

```
<!---
<fusedoc fuse="qryGetEmployees.cfm" language="ColdFusion"
version="2.0">
  <responsibilities>
    I return a recordset of employee data.
  </responsibilities>
  <io>
    <out>
      <recordset name="qryGetEmployees">
        <number name="employeeID" />
        <string name="firstName" />
        <string name="lastName" />
        <string name="department" />
        <number name="supervisorID" />
      </recordset>
    </out>
  </io>
</fusedoc>
--->
<cf_querysim>
  qryGetEmployees
  employeeID,firstName,lastName,department,supervisorID
  8|Helen Hotel|ENGINEERING|2
  2|Belinda|Baker|MANAGEMENT|0
  4|Deborah|Delta|MARKETING|2
  5|Eugene|Echo|MARKETING|4
  6|Francine|Foxtrot|MARKETING|4
  7|Gary|Golf|MARKETING|4
  1|Andy|Able|SALES|3
  3|Charlie|Charlie|SALES|2
</cf_querysim>
```

Listing 5: Using <cf_querysim> to simplify create/edit forms

```
<!---
<fusedoc fuse="dspUserForm.cfm" language="ColdFusion" spec-
ification="2.0">
  <responsibilities>
    I display a for to edit a user's record.
  </responsibilities>
  <io>
    <in>
    <recordset name="qryGetUser" comments="If empty, create
querysim with a blank row">
      <string name="email" />
        <string name="firstName" />
        <string name="lastname" />
    </recordset>
  </in>
  <out>
    <string name="email" scope="form" />
      <string name="firstName" scope="form" />
      <string name="lastname" scope="form" />
  </out>
  </io>
</fusedoc>
--->
<cfif not qryGetUser.recordCount>
  <cf_querysim>
  qryGetUser
  email,firstName,lastName
  null|null|null
  </cf_querysim>
</cfif>
<form action="#self#?fuseaction=updateUser" method="post">
  <cfoutput query="qryGetUser">
  E-mail: <input type="text" name="email"
value="#qryGetUser.email#"><br>
  First Name: <input type="text" name="firstName"
value="#qryGetUser.firstName#"><br>
  Last Name" <input type="text" name="lastname"
value="#qryGetUser.lastName#"><br>
  <input type="submit">
</form>
```

Listing 6: Querysim for WorkLog

```
<!---
<fusedoc fuse="qryGetWorkLog.cfm" language="ColdFusion"
version="2.0">
  <responsibilities>
    I return a recordset of work log entries.
  </responsibilities>
  <io>
    <out>
      <recordset name="qryGetWorkLog">
        <datetime name="itemDate" />
        <string name="itemText" />
      </recordset>
    </out>
  </io>
</fusedoc>
--->
<cf_querysim>
  qryGetWorkLog
  itemDate,itemText
  25 Mar 02|Created new training table
  26 Mar 02|Worked on user training edits
  26 Mar 02|Inserted an alert message in the onload event
of the form if the qry resulted in no records
  26 Mar 02|Checked validation requirements on forms to
ensure that at least one field must be filled in before
submitting
  26 Mar 02|Conducted training session
  26 Mar 02|Developed Oracle SQL queries and procedures
for the Widgets circuit
  26 Mar 02|Modified the Widgets module to show line items
in a grouped listing
  26 Mar 02|Tested changes to Widgets module
  26 Mar 02|Prepared matrix of user roles/functions
</cf_querysim>
```


Listing 7: Display file for WorkLog

```
<!---
<fusedoc fuse="dspWorkLog.cfm" language="ColdFusion" ver-
sion="2.0">
  <responsibilities>
    I display work log entries by date.
  </responsibilities>
  <io>
    <out>
      <recordset name="qryGetWorkLog">
        <datetime name="itemDate" />
        <string name="itemText" />
      </recordset>
    </out>
  </io>
</fusedoc>
--->

<link rel="stylesheet" href="WorkLog.css" type="text/css">
<title>Work Log</title>
<body bgcolor="#FFFFCC" text="#006600">
<table width="580" border="0" align="center">
  <tr>
    <td>
      <div align="center" class="pageTitle">Work Log</div>
    </td>
  </tr>
  <tr>
    <td> <cfoutput query="qryGetWorkLog" group="itemDate">
        <span class="itemDate">#itemDate#</span><br>
        <cfoutput> <span class="itemText">-
#itemText#</span><br>
  </cfoutput>
   </cfoutput>
 </td>
  </tr>
  <tr>
    <td> </td>
  </tr>
</table>
```

Listing 8: Querysim with <cfinclude>

```
<!---
<fusedoc fuse="qryGetWorkLog.cfm" language="ColdFusion"
version="2.0">
  <responsibilities>
    I return a recordset of work log entries.
  </responsibilities>
  <io>
    <out>
      <recordset name="qryGetWorkLog">
        <datetime name="itemDate" />
        <string name="itemText" />
      </recordset>
    </out>
  </io>
</fusedoc>
--->

<cf_querysim>
  qryGetWorkLog
  itemDate,itemText
  <cfinclude template="WorkLog.txt">
</cf_querysim>
```

Listing 9: WorkLog text file

```
25 Mar 02|Created new training table
26 Mar 02|Worked on user training edits
26 Mar 02|Inserted an alert message in the onload event of
the form if the qry resulted in no records
26 Mar 02|Checked validation requirements on forms to
ensure that at least one field must be filled in before
submitting
26 Mar 02|Conducted training session
26 Mar 02|Developed Oracle SQL queries and procedures for
the Widgets circuit
26 Mar 02|Modified the Widgets module to show line items
in a grouped listing
26 Mar 02|Tested changes to Widgets module
26 Mar 02|Prepared matrix of user roles/functions
```

Listing 10 : Customized Apache access.log

```
127.0.0.1|-|-|[26/Mar/2002:13:56:56 -0500]|"GET
/test HTTP/1.0"|301|292
127.0.0.1|-|-|[26/Mar/2002:13:56:56 -0500]|"GET
/test/ HTTP/1.0"|200|4286
```

```
127.0.0.1|-|-|[26/Mar/2002:13:56:57 -0500]|"GET
/icons/unknown.gif HTTP/1.0"|200|245
127.0.0.1|-|-|[26/Mar/2002:13:56:57 -0500]|"GET
/icons/image2.gif HTTP/1.0"|200|309
127.0.0.1|-|-|[26/Mar/2002:13:56:57 -0500]|"GET
/icons/folder.gif HTTP/1.0"|200|225
127.0.0.1|-|-|[26/Mar/2002:13:56:57 -0500]|"GET
/icons/blank.gif HTTP/1.0"|200|148
127.0.0.1|-|-|[26/Mar/2002:13:56:57 -0500]|"GET
/icons/text.gif HTTP/1.0"|200|229
127.0.0.1|-|-|[26/Mar/2002:13:57:09 -0500]|"GET
/test/Image1.gif HTTP/1.0"|200|805
127.0.0.1|-|-|[26/Mar/2002:13:57:14 -0500]|"GET
/test/addTest.html HTTP/1.0"|200|242
```

Listing 11: WebLog querysim file

```
<!---
<fusedoc fuse="qryGetApacheLog.cfm" language="ColdFusion" ver-
sion="2.0">
  <responsibilities>
    I return a recordset of Apache access.log entries.
  </responsibilities>
  <io>
    <out>
      <recordset name="qryGetApacheLog">
        <string name="client_address" />
        <string name="userid" />
        <string name="request_time" />
        <string name="request" />
        <string name="response_status" />
        <string name="size" />
      </recordset>
    </out>
  </io>
</fusedoc>
--->

<cf_querysim>
  qryGetLogs
  client_address,ident,userid,request_time,request,response_sta-
tus,size
  <cfinclude template="access.log">
</cf_querysim>
```

Listing 12: WebLog display file

```
<!---
<fusedoc fuse="dspApacheLog.cfm" language="ColdFusion" ver-
sion="2.0">
  <responsibilities>
    I display an Apache access.log report.
  </responsibilities>
  <io>
    <in>
      <recordset name="qryGetApacheLog">
        <string name="client_address" />
        <string name="userid" />
        <string name="request_time" />
        <string name="request" />
        <string name="response_status" />
        <string name="size" />
      </recordset>
    </in>
  </io>
</fusedoc>
--->

<link rel="stylesheet" href="WorkLog.css" type="text/css">
<title>Apache access.log</title>
<body bgcolor="#FFFFCC" text="#006600">
<table width="580" border="0" align="center">
  <tr>
    <td>
      <div align="center" class="pageTitle">Apache access.log</div>
    </td>
  </tr>
  <tr>
    <td>
    <cfdump var="#qryGetApacheLog#">
 </td>
  </tr>
  <tr>
    <td> </td>
  </tr>
</table>
```

Listing 13: Calling file for WebLog

```
<cfinclude template="qryGetApacheLog.cfm">
<cfinclude template="dspApacheLog.cfm">
```

CODE LISTING

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

ColdFusion MX and XML: **Creating XML** Part 2

Export your data to any XML language with relative ease

BY **DAVID GASSNER**

I'll plead guilty to having tortured more than my share of bytes into completely unnatural configurations.

In the BX years (Before XML), if you had to share data with your business partners you might have represented the data in comma-delimited text files. If the data was too complex to fit one record to a line, you might have created a proprietary design to hold the information.

Today you'd probably consider XML. Its consistency makes it possible to create and parse text-based data files with standardized programming tools. When deciding how to transfer your data to XML you'd have a choice of programming approaches. Some would be hard, others easier, but until now none would have been native ColdFusion.

That's changed with the release of ColdFusion MX. This is the second of three articles describing the new XML development tools in CFMX. In the first article (***CFDJ***, Vol. 4, issue 6) I described methods for parsing, extracting, and searching for data from XML. This month I'll describe the new tools for turning your data into XML.

Toto, I Don't Think We're in WDDX Anymore

When I speak of XML, I'm not just talking about Web Distributed Data Exchange (WDDX), which has been supported in ColdFusion since version 4. While WDDX is a wonderful XML language, it hasn't been adopted widely beyond the world of Macromedia. The rest of the computing world uses other variants of XML, and there are too many different flavors to count. You need to be able to publish XML in any format, not just WDDX. That's what these tools provide.

Imagine that I have a database table containing employee information that I want to move to XML. In this example I'll use the employees table from the CFSnippets database that's installed with CFMX.

CFMX offers two methods for creating and publishing an XML document:

- The "Text" method uses the new <CFXML> tag and allows you to model the XML tree as text, much as you use <CFOutput> to model HTML output.
- The "Object" method uses CFMX's new XML functions to build an object model.

The Text method is considerably easier to use, but I'll show both approaches and you can decide for yourself.

Modeling XML as Text

The Text method builds the XML structure as text, then converts it to an XML document object variable. First, create a model of the XML structure inside a pair of <CFXML> tags. Use placeholders to indicate where the variable data will go:

```
<CFXML variable="xmlDoc">
  <Employees>
    <Employee ID="XXX">…Child
Elements…</Employee>
  </Employee>
</CFXML>
```

The <CFXML> tag creates a new XML document object variable named *xmlDoc*. As I described in the last article, you can view the structure of the resulting XML document object with the <CFDump> command:

```
<CFDump var="#xmlDoc#">
```

See Figure 1 for the dumped structure of the XML document before being populated with real data.

Adding Your Data

Assuming your content is in a database, add a <CFQuery> before the call to <CFXML>. To add variable content to the XML document, wrap the employee element with a <CFOutput> tag pair that loops over the query contents. This is just like looping over a query to create multiple list items or table rows in HTML:

```
<CFOutput query="qEmps">
  <Employee>…</Employee>
</CFOutput>
```

To place the data where you want it, replace the placeholders with the data from the query. If there might be any reserved characters in a value, such as ampersands (&), quotes (“), or tag symbols (“<” or “>”), wrap the value with the XMLFormat() function. This replaces any such characters with their equivalent entities and ensures that your XML is well formed:

```
<Employee ID="#qEmps.Emp_ID#">
<FirstName>#XMLFormat(qEmps.Firs
tName)#</FirstName>
</Employee>
```

Notice that the element tags are wrapped around the value without any additional white space (spaces, tabs, line feeds). Adding such space for readability is fine when creating HTML, since the Web browser usually normalizes the resulting output, removing extra spaces. In XML you'd be changing the data.

Your XML document object has been created. Once again, you can view the structure of the document with <CFDump>. See Listing 1 for the complete code, and Listing 2 for the contents of the XML file it creates.

The toString() Function

Right now your XML document is a structured object that you can read programmatically using the techniques described in last month's article. To share the data, you now need to convert the document to text. For this, wrap the XML document object variable in the toString() function. For instance, to publish the XML document to a text file, use this code:

```
<CFFile action="WRITE"
  file="#ExpandPath('.')#\employees.xml"
  output="#toString(xmlDoc)#">
```

The toString() function adds an XML declaration at the beginning of the output and always sets the declaration's “encoding” attribute to UTF-8.

Modeling XML as Objects

The Object method of creating XML files uses two new XML functions:

- XMLNew(): Creates a new XML document
- XMLElemNew(): Creates a new XML element

Start with XMLNew(). This function returns an instance of an empty XML document:

```
<CFSet xmlDoc=XMLNew()>
```

The document initially isn't well formed – that is, it doesn't contain any elements. So next you'll create a new element as a child of the document. XMLElemNew() takes two arguments. The first is the XML document object to which you're adding the element. The second is the name of the new element.

The document has a built-in child object named *xmlRoot*. By assigning a new element to the xmlRoot, you're creating the document's root element:

```
<CFSet xmlDoc.xmlRoot=
  xmlElemNew(xmlDoc, "Employees")>
<CFSet root=xmlDoc.XMLRoot>
```

Now, loop through the query just as before, but this time create elements to represent each employee. Each element contains a built-in array named XMLChildren, and each item of the array is a child element. Adding a new employee element to the root looks like this:

```
<CFSet ArrayAppend( root.xmlChildren,
  xmlElemNew(xmlDoc, "Employee") )>
```

As the XML hierarchy gets deeper, the syntax for referring to any particular element can get unwieldy. So now create a reference variable that points to the element you just created:

```
<CFSet emp=root.xmlChildren[
  ArrayLen(root.xmlChildren)]>
```

Each employee has an ID that I'd like to store as an XML attribute. Each element has a built-in structure

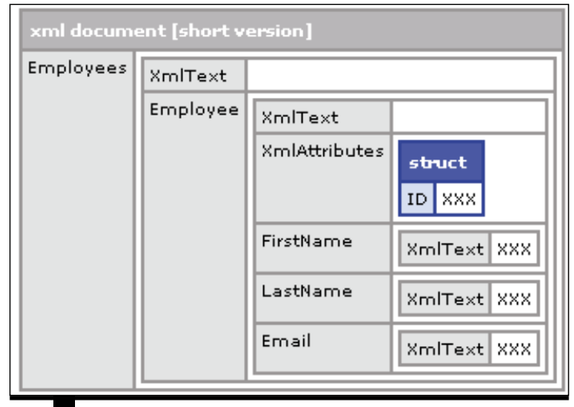


Figure 1 XML document without data

called XMLAttributes. When you assign an item to the structure, you're adding an attribute to the element:

```
<CFSet emp.xmlAttributes.ID = qEmps.Emp_ID>
```

Each employee has a set of values that I want to store as child elements with text values between the tags. Once again I'll use XMLElemNew() to create the element, then set the new element's built-in XMLText property to the value from the query. I don't have to use the XMLFormat() function to wrap the data as the Object method replaces any reserved characters for me:

```
<CFSet ArrayAppend( emp.xmlChildren,
  xmlElemNew(xmlDoc, "FirstName"))>
<CFSet emp.FirstName.xmlText=
  qEmps.FirstName>
```

Repeat this step for each of the child elements you want to add and populate with data. When you run the code you'll have created an XML document object containing your data.

This time, instead of saving to disk, I'll share the XML over the HTTP connection. Before outputting to the browser, I'll use <CFContent> to set the content type to XML. This lets any users of the data know that they're getting the right type of information:

```
<CFContent type="text/xml">
```

The code for the Object method is in Listing 3.

White-Space Issues

You may notice that the <CFXML> version of the output has a little too much white space between employee elements. This is okay as an XML parser doesn't make any distinction between a little bit of white space and a lot of white space.

The Object version, though, comes out in a single line and is difficult for the human eye to read:

```
<Employees><Employee ID="1"><FirstName>…
```

I fix this with a call to a user-defined function I wrote named XMLIndent() (stored in a file named *xmlUtilities.cfm*). See Listing 4 for its contents.

XMLIndent() passes the XML document through an XSLT (Extensible Stylesheet Language Transformation) template that returns the XML document as is, but adds hard returns to make it more readable. XMLIndent() takes the XML document as an argument and returns a text representation of the indented result:

```
<CFInclude template="xmlUtilities.cfm">
<CFSet xmlText=xmlIndent(xmlDoc)>
```

I'll discuss this UDF more thoroughly in the next article, which describes what you can do with ColdFusion and XSLT.

Limitations

Both methods of creating XML files have these limitations:

- ***UTF-8 encoding only***: While the toString() function has an "encoding" attribute, it doesn't work for the XML document object. And while CFFile now has a "charset" attribute to output in other encoding formats, it doesn't affect the resulting XML declaration's encoding attribute. You can easily end up with a file in UTF-16 format, but an XML declaration that claims the file is in UTF-8.
- ***No Document Type Declarations***: There is no built-in method for assigning a Document Type Definition (DTD) to the XML output. Adding the DTD to the XML text in <CFXML> doesn't throw an error, but it's also ignored.

I'll describe workarounds for both of these problems in the next article. (If you need either of them right away, e-mail me.)

Conclusion

I've described two methods of creating XML files with ColdFusion MX. The Text method uses <CFXML> and will seem most intuitive to experienced ColdFusion developers as it allows you to create XML with the same simple methods we use to create HTML dynamically. The Object method gives you greater control over the structure of your XML document at the expense of greater coding complexity. Either method allows you to export your data to any XML language with relative ease.

• • •

In the last article of this series I'll describe the use of XSLT and the new XMLTransform() function, including how to use XSLT to move your data from WDDX to other flavors of XML and back again.

ABOUT THE AUTHOR

David Gassner is a Macromedia-certified instructor and the author of an upcoming course on XML development for ColdFusion developers. He teaches classes in ColdFusion, HTML, Java, and XML as well as other Web development skills.

@ DGASSNER@NAPANET.NET

Listing 1: TextMethod.cfm

```
<!--- Querying and moving data from a database
  to XML - the Text method --->

<cfquery name="qEmps" datasource="CFSnippets">
select * from Employees
</cfquery>

<cfxml variable="xmlDoc">
<Employees>
  <cfoutput query="qEmps">
  <Employee ID="#qEmps.Emp_ID#">
    <FirstName>#qEmps.FirstName#</FirstName>
    <LastName>#qEmps.LastName#</LastName>
    <Email>#qEmps.Email#</Email>
  </Employee>
  </cfoutput>
</Employees>
</cfxml>

<cfdump var="#xmlDoc#">
<cffile action="WRITE"
  file="#ExpandPath('.')#\employees.xml"
  output="#toString(xmlDoc)#">
```

Listing 2: Employees.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Employees>

  <Employee ID="1">
    <FirstName>Carolynn</FirstName>
    <LastName>Peterson</LastName>
    <Email>CPETERSON</Email>
  </Employee>

  <Employee ID="2">
    <FirstName>Dave</FirstName>
    <LastName>Heartsdale</LastName>
    <Email>FHEARTSDALE</Email>
  </Employee>

  <Employee ID="3">
    <FirstName>Linda</FirstName>
    <LastName>Stewart</LastName>
    <Email>LSTEWART</Email>
  </Employee>

  ... more Employee elements ...

</Employees>
```

Listing 3: ObjectMethod.cfm

```
<!--- Querying and moving data from a database
  to XML - the Object method --->

<cfquery name="qEmps" datasource="CFSnippets">
select * from Employees
</cfquery>

<!--- Create a new XML document --->
<cfset xmlDoc=XMLNew()>
<cfset xmlDoc.XMLRoot=
  xmlElemNew(xmlDoc, "Employees")>
<cfset root=xmlDoc.XMLRoot>

<cfloop query="qEmps">
  <!--- Create a new element and get a reference to it --->
  <cfset ArrayAppend(root.xmlChildren,
    xmlElemNew(xmlDoc, "Employee") )>
  <cfset emp=root.xmlChildren[
    ArrayLen(root.xmlChildren)]>

  <!--- Set an attribute --->
  <cfset emp.xmlAttributes.ID = qEmps.Emp_ID>

  <!--- Add child elements and populate with data --->
  <cfset ArrayAppend(emp.xmlChildren,
    xmlElemNew(xmlDoc, "FirstName"))>
  <cfset emp.FirstName.xmlText=qEmps.FirstName>
  <cfset ArrayAppend(emp.xmlChildren,
    xmlElemNew(xmlDoc, "LastName"))>
  <cfset emp.LastName.xmlText=qEmps.LastName>
  <cfset ArrayAppend(emp.xmlChildren,
    xmlElemNew(xmlDoc, "Email"))>
  <cfset emp.Email.xmlText=qEmps.Email>
</cfloop>

<!--- Include the utilities file --->
<cfinclude template="xmlUtilities.cfm">

<!--- Output the resulting XML to the browser --->
<cfcontent type="text/xml">
<cfoutput>#XMLIndent(xmlDoc)#</cfoutput>
```

Listing 4: xmlUtilities.cfm

```
<!--- xmlUtilities.cfm - functions to
  help with XML processing --->

<!--- XMLIndent()
  requires one argument of type XML
  returns the resulting XML as text --->
<cffunction name="XMLIndent" returntype="string">
 <cfargument name="xml" type="any" required="Yes">

 <cfset xsl='<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output indent="yes"/>
    <xsl:template match="/">
     <xsl:copy-of select="."/>
    </xsl:template>
  </xsl:stylesheet>'>
 <cfset output=xmltransform(
  tostring(arguments.xml), variables.xsl)>
 <cfreturn output>

</cffunction>
```

CODE LISTING

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

Beyond CFMAIL

Sending large volumes of e-mail

BY TOM PEER

One of the reasons I was first drawn to ColdFusion was the built-in functionality for such things as sending e-mail, making HTTP requests, FTP uploads – all the myriad subsidiary functions you inevitably find yourself using when you build and manage large sites.

Content has to be downloaded from here, uploaded to there, and e-mailed to thousands of users every day.

Using only standard ColdFusion functionality keeps things neat and easy to support. And for all the warnings about not running an SMTP server on the same machine as your Web server, I'm sure many developers have a site running somewhere that's doing exactly that and working extremely well. I also suspect many developers have pushed things a little too far and found that nothing soaks up resources quite like an overloaded mail server.

Even if you do run an SMTP server on a separate machine, large volumes of e-mail can still have a catastrophic effect on performance. CFMAIL is a great tag and, unlike other third-party tags such as CFFTP or CFPOP, it probably won't result in a "hanging thread." However, CFMAIL writes only text files to a spool, which then has to be sent to an SMTP server, and this spooling uses up resources. There comes a point when you not only want the SMTP service to be running on a separate machine, but the spooling as well.

There's a lot of discussion about the problems with CFMAIL in the support forums, but almost all the problems are not due to the CFMAIL tag, which works very well, but to this secondary spooling process. It's this process that leaks file handles that can eventually bring the ColdFusion service to a stop (fixed in version 5), and it can also fail when you specify different SMTP servers in the same batch of mails.

As good as CFMAIL is, there are several occasions when you need functionality that it doesn't have. First, it can't prioritize mail, so if someone requests a password reminder while your system is sending out 2,000 newsletters, they'll have to wait. Second, it can send only text or HTML mail without the option of a text "failover" for HTML. Third, it can't embed images as inline MIME attachments.

This means that people who download their mail before reading offline won't see the image or might be annoyingly prompted to reconnect. Whether or not this is a serious problem is one of those difficult calls that has no correct answer. If you want everyone to see your mail perfectly, you need to attach inline images. If you're confident that 99% of your subscribers read your mail at corporate workstations, then spread your bandwidth load and possibly track e-mail reads by using remote images.

The lack of a text failover is a more serious problem. Although almost all mail clients can now handle HTML, some people choose to turn this functionality off. Including a "Can't read this?" link at the top of the page is no real substitute for a text alternative.

There are commercial alternatives to CFMAIL (COM and Java), but there's a simple and quick way to get around these issues. It's relatively straightforward to bypass CFMAIL and write your e-mails directly to your mail server's outbound queue folder. This allows you to use inline images and failovers, but far more important, also allows you to separate Web serving and mail spooling applications by writing mail first to a database and then to a mail queue.

That might sound like a daunting prospect, but a basic e-mail couldn't be much simpler. It needs only from, to, and subject fields and some body copy. Try saving a file like this to your mail server's outgoing queue folder.

```
From: tom <tom@tompeerconsult-
ing.com>
To: test <test@tompeerconsult-
ing.com>
Subject: Testing

Body text goes here
```

For the Microsoft SMTP service, this is a folder called "pickup" within the mail root (usually C:\inetpub\mailroot). Bad mails are moved to "badmail" in the same folder. If you use the ever popular MDaemon mail server (www.deerfield.com), save to the REMOTEQ folder or to a custom remote queue folder you've set up. Many other systems work in exactly the same way – you just have to find the location of the outbound queue.

If you're not interested in HTML and inline images, you already have your solution to bulk mailing – simply writing text files like this to an outgoing queue folder. It's easily scalable to more than one server by either writing to several servers from a central "broker" or else running code on each mail server to fetch batches of mail from a database at regular intervals. The former is easier (and probably cheaper); the latter more powerful and necessary when you're sending huge volumes of mail (see Figure 1).

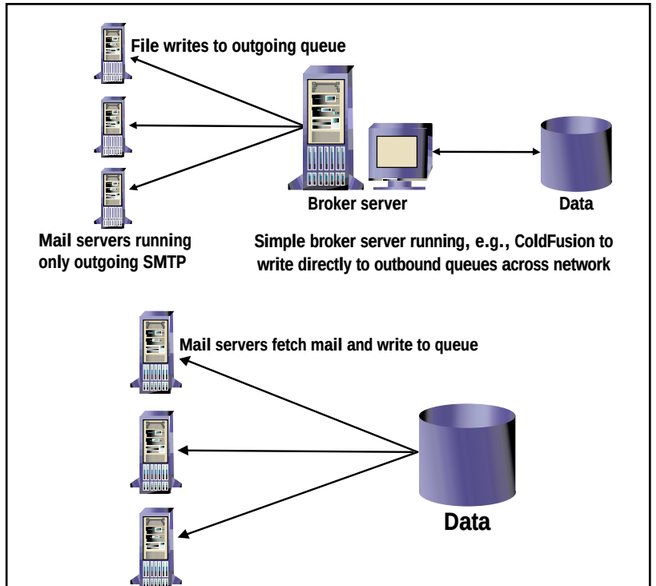


Figure 1 Possible setups for sending mail from multiple SMTP servers

Using the simpler first approach you can probably avoid keeping a permanent record in the database of the e-mails you're sending. For instance, if you have a newsletter to send to a list of subscribers in a table called USERS, you could select every row of USERS from the database, loop over it, and write a text file for each e-mail.

Suppose, though, something goes wrong mid-mailing. If a mail server or the broker machine fails, how do you pick up where you left off? You can either send the whole mailing again or leave out the people that didn't get it.

A better arrangement is some sort of permanent record; for example, a table of mailings in which the EMAIL_STATUS_ID indicates whether an e-mail has been sent (see Figure 2). Alternatively, you can omit this field and delete from the EMAIL_RECIPIENTS table when the mail has been sent. Using SQL Server you can then insert into the "queue" with a procedure like this:

```
CREATE PROCEDURE SEND_NEWSLET-
TER
(@EMAIL_ID INT)
AS INSERT INTO EMAIL_RECIPIENTS
(USERS_ID, EMAIL_ID, EMAIL_STA-
TUS_ID)
SELECT USERS_ID, @EMAIL_ID, 1
FROM USERS
```

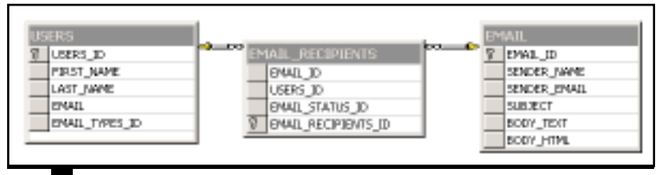


Figure 2 A simple schema for sending bulk mails, e.g., a newsletter

and fetch the mail in batches with another procedure (see Listing 1).

This allows you to use the database as a broker and keeps the level of functionality needed for the mail servers in the second scenario to a minimum – a quick substitution of the important database fields (e.g., e-mail, first_name, last_name) and a file write to the outbound queue.

If you want to send out more than simple text messages, either immerse yourself in the RFC-822 standard or else use one of the solutions available in the Developers' Exchange. CF_ADVANCEDEMAIL by Jochem van Dieten is a great tag; it takes care of text failovers and inline images, but you'll need to give it a couple of tweaks if you want to use it to write directly to an outgoing queue and not the ColdFusion spool (an annotated version is available from my Web site at www.tompeerconsulting.com). The syntax is straightforward (see Listing 2), and creating inline images couldn't be simpler. Just embed them with full URLs in your e-mail and the tag will download, encode, and attach them to the mail. Rather than call this tag every time you write a text file for a bulk mailing, a simple customization will allow you to call it once and then use the formatted result as a template for writing future e-mails.

If the problem of mail priorities is an issue on your site, add a priority field to your database table of e-mails, then write higher-priority mails to a dedicated server or a separate SMTP instance running on a different port. You may even decide to write high-priority mails to an MDaemon or equivalent mail server queue, where they can be monitored, and send less important e-mails directly via SMTP.

If, like me, you develop on Windows but rather envy Linux users their price-to-performance ratios, by separating mail sending from Web serving you can quite happily use different platforms for the two tasks. A well-specified Windows server might struggle to send out 3,000 mails an hour while a Linux system costing half as much running just Sendmail might send out 5,000 or 6,000 an hour.

Whatever platform or e-mail server you choose, keeping your mail spooling separate will enable you to scale your application simply by adding more servers, especially if you opt for the second scenario described earlier. On one project we were able to send out about 250,000 e-mails a day from three discarded office PCs running Red Hat Linux – saving thousands from a third-party contract – while operating under a budget freeze.

Finally some words of warning before you get started. Make sure you secure your SMTP servers against relaying. Be careful you don't use up all your bandwidth sending outgoing mail. Do test your mail on clients like Eudora, Netscape, and Lotus – there are more older e-mail clients than older browsers, and Outlook tolerates a great deal that they don't. If you're sending bulk mailings, make sure you comply with all relevant legislation, include an unsubscribe mechanism (URL or e-mail address), and make it clear how you came by the subscriber's details.

@TOM@TOMPEERCONSULTING.COM

ABOUT THE AUTHOR

Tom Peer specializes in migrating printed publications online and has recently completed the online edition of The World Handbook of Stock Exchanges *(www.exchangehandbook.com). He has worked in electronic publishing for eight years, including a stint as the manager of New Scientist Online (www.newscientist.com)*

Listing 1

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRANSACTION
SELECT TOP 500 * FROM EMAIL_VIEW
WHERE EMAIL_STATUS_ID = 1 ORDER BY EMAIL_ID, USERS_ID
UPDATE EMAIL_RECIPIENTS SET EMAIL_STATUS_ID = 1 WHERE
EMAIL_RECIPIENTS_ID IN
(SELECT TOP 500 EMAIL_RECIPIENTS_ID FROM EMAIL_RECIPIENTS
WHERE
EMAIL_STATUS_ID = 1 ORDER BY EMAIL_ID, USERS_ID)
COMMIT
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

Listing 2:cf_advancedemail syntax

```
<cf_advancedemail from="tom <tom@tompeerconsulting.com>"
to="test <test@tompeerconsulting.com>"
 format="TEXT"
 tempdir="C:\inetpub\mailroot\"
 spooldir="C:\inetpub\mailroot\pickup\"
 type="ALTERNATIVE"
 mode="SILENT">
 <cf_advancedemailparam contenttype="TEXT">
 Text version here
</cf_advancedemailparam>

<cf_advancedemailparam name="subject" value="SUBJECT">

<html>
<head>
 <title>Testing mail</title>
</head>

<body>
<p>
<b>Here's the HTML</b>
</p>
</body>
</html>

</cf_advancedemail>
```

CODE LISTING

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

Beyond <CFQUERY> Part 2 of 2

Singing the praises of stored procedures and sequences

BY **NEIL ROBERTS**

This month I hope to shed some light on Oracle's larger datatypes for those of you who like to do things big.

Let's go straight to the deep end with stored procedures. They're faster and more secure than standalone queries. In the first instance, procedures are faster because a lot of the execution process has already been completed before runtime, as I mentioned in my previous article(***CFDJ***, Vol. 4, issue 6) (In that article I covered some of the many ways you can use Oracle with the rather humble-looking <CFQUERY> tag.) The reason is that the validation process is done at compilation time, so when you invoke the procedure from your ColdFusion template, all it has to do is run.

Obviously, there are more generic benefits to using stored procedures, as they provide a degree of separation from your ColdFusion code, meaning that as long as the procedure call doesn't change, it's entirely possible to update the Oracle procedure incorporating database changes without having to alter your front-end ColdFusion application at all. Also, as I mentioned in my last article, using functions and procedures allows you to share the load across servers (provided you have separate servers for your ColdFusion and Oracle installations, as Macromedia recommends).

Procedures have the added benefit of hiding some of the complexity of database interaction. As is quite often the case, there's a mix of skills in any development team, and using procedures means that your strong Oracle developers can develop the database application, your ColdFusion developers can create the front end, and neither will have to have an in-depth knowledge of what the other is doing.

Some of you may recall a recent article by Ben Forta on <CFQUERYPARAM> ("Faster and Safer Database Queries," ***CFDJ***, Vol. 4, issue 2) in which he highlighted the added security it provided through facilitating datatype checking. The same is true for calling stored procedures, which requires the use of <CFPROCPARAM> instead (see Listing 1). This also extends to the Oracle side of the application, where the procedure has to define the datatypes (see Listing 2). Combine this with proper exception handling and you have more secure database access. The two listings constitute an example of a simple Oracle procedure, showing parameter declaration basic exception handling.

The example procedure includes two types of calls to sequences (seq_category, seq_img_id). As you can see, they are simple to use and just as easy to create:

```
/* this is a simple sequence
creation script*/
CREATE SEQUENCE seq_example
increment by 1 start with 1;
```

And they ensure that all your primary keys are unique to boot. The only reason I mention this at all is that in the not too distant past I recall inheriting some code in which the developer had created a primary key by selecting the Max(id) from the table and then adding 1 to it. The main issue with this is that if someone else is running the same page at the same time, you're going to retrieve the same Max(id) and therefore, ultimately, try to insert a record with the same primary key.

How Much Can a CLOB Hold?

Recently I followed a thread on the Macromedia Forums discussing the use of CLOBS in Oracle–ColdFusion applications. The crux of the issue was that no one seemed to know how to fully utilize the 4GB of storage space Oracle says a CLOB can hold. As the thread progressed, it emerged that a number of people were unable to input data held in a CLOB that exceeded 4,000 characters. Reading on, I saw a range of possible solutions using Oracle's dbms_lob package, but as ColdFusion cfstoredproc doesn't support CLOBs, it seemed this couldn't be the best solution. I decided to investigate the problem. Initial searches proved fruitless, so I ran a few experiments. True to form, I found I could successfully input 4,000 characters to a CLOB, but as soon as I tried to put in 4,001, I got this error:

```
Oracle Error Code = 1704

ORA-01704: string literal too long
```

Now my math isn't too hot, but that's a far cry from the 4GB Oracle had promised. So I looked still deeper and discovered a thread on a disparate forum located in one of the far corners of the Web where some helpful individual hinted that there were certain limitations with Oracle native drivers. With this is mind, I recalled a time when I dabbled in some light Delphi development that introduced me to ODBC drivers. Now it's widely accepted that ODBC drivers are slower than native drivers, but in the course of my experiment I discovered that this time lag might well be worth the wait. I got our DBA to install the Merant ODBC drivers and set up a datasource pointing to the database containing my CLOB table.

I then ran the following code, increasing the loop size to see what would happen:

```
<cftry>

<cfset v_txt = "start">
<cfloop from="1" to="1000">
  <cfset v_txt = v_txt &
"0123456789">
</cfloop>
<cfset v_txt = v_txt & "end">

<cfquery name="clob_test" data-
source="#request.ODBC_ds#">
  INSERT INTO clob_test VALUES
(seq_clob.nextval, '#v_txt#')
</cfquery>

<cfcatch type="DATABASE">

<cfoutput>#cfcatch.message#</cfo
utput>
</cfcatch>
</cftry>
```

The result (as you'll find if you try it yourself) is way beyond the measly 4,000 characters facilitated by the native drivers. In fact, I consistently managed to increase this by four times – to just over 16,000 characters. Admittedly this is a long way short of the 4GB promised by Oracle, but in reality it's plenty to store dynamic content in.

The story doesn't end there, however. Late in my testing I stumbled across a new cf_sql_datatype that relates back to what I mentioned at the beginning of this article, where <CFQUERYPARAM> allows you to do datatype checking. Now I'm familiar with the normal array of varchar, number, and so on, but I'd never seen cf_sql_clob, so I tried this out by using the trusty <CFQUERYPARAM> tag in my insert statement and specifying the datatype as cf_sql_clob, making the insert look something like this:

```
  <cfquery name="clob_test"
   datasource="#attributes.ds#">
 INSERT INTO clob_test
    VALUES (seq_clob.nextval,
        <cfqueryparam cfsql
        type="CF_SQL_CLOB"
        value="#v_txt#">)
  </cfquery>
```

Using Oracle native drivers I got a number of errors. On 8.1.5 I got this one:

```
Oracle Error Code = 1461

ORA-01461: can bind a LONG value only for
insert into a LONG column
```

And on 8.1.6:

```
Oracle Error Code = 936

ORA-00936: missing expression
```

Not good. However, Merant came to the rescue, and I now have no problem inserting into my CLOBs. I actually let it stand at a final figure of around 400,000 characters successfully inserted.

The Driver Makes the Difference

So there you have it. The trick is not in complex Oracle coding, but in selecting the right drivers. If you're really worried about the time lags associated with ODBC drivers, then use them only for your CLOB manipulation. The main thing is, you won't have to change your code at all, but be sure your datasource is configured properly in the ColdFusion Administrator, either by enabling Long text retrieval or setting the buffer to an adequate level.

ABOUT THE AUTHOR

Neil Roberts is an analyst programmer with a large international law firm based in London. His focus is on developing a range of Web sites and Web-based applications using both ColdFusion and Oracle. He is also an active member of the UK ColdFusion user group.

Oracle also offers another LOB in the form of binary large objects (BLOBs). As the name suggests, these store the information in binary format. It's possible to store all manner of information in your database: images, word documents, test files, even PDFs – the list is nearly endless.

Don't get too excited, though. While Oracle does provide the handy dbms_lob package with which you can insert BLOBs, check their length, see if they're empty, and so on, it's annoying that you can retrieve them only in binary format. Now I don't know about your users, but I've found mine prefer that the Word document they stored in the database comes out of the database as a Word document. Unfortunately, when you simply select a BLOB from the database…well, it comes out in binary. So this is a bit of a problem, as I'm sure you can appreciate.

All is not lost, however. You can create your own CFX tag to apply the mime type to the binary and format it correctly. Unfortunately, this is beyond the scope of this article, as my Java knowledge is minimal and my C++ nonexistent, but I won't leave you high and dry. There are two solutions for those whose pockets are deeper than their Java knowledge. There's a CFX tag called BLOB manager, available from the Macromedia tag library. I've tried the online demo, which seems pretty slick. For the more Java-savvy or less well funded, try out the tutorial at www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=45&lngWId=9.

• • •

Well, that's all from me, folks, but remember: use sequences for your primary keys and stored procedures to hide complexity, and be wary when handling large objects.

@NSROBERTS@HOTMAIL.COM

Listing 1: ColdFusion call to the procedure

```
<cfstoredproc procedure="prc_ins_data"
datasource="#request.ds#">
 <cfprocparam type="OUT"
 cfsqltype="CF_SQl_NUMERIC"
 variable="v_img_id" dbvarname="p_id" value ="">
 <cfprocparam type="IN"
 cfsqltype="CF_SQl_VARCHAR"
 variable="v_img_title" dbvarname="p_img_title"
 value ="#form.title#">
 <cfprocparam type="IN"
 cfsqltype="CF_SQl_VARCHAR"
 variable="v_img_desc" dbvarname="p_img_desc"
 value ="#form.description#">
 <cfprocparam type="IN"
  cfsqltype="CF_SQl_VARCHAR"
  variable="v_img_categories" dbvarname="p_categories"
  value ="#form.categories#">
</cfstoredproc>
```

Listing 2: Oracle code

```
CREATE OR REPLACE PROCEDURE prc_ins_data(p_id  IN OUT INTE-
GER,
    p_img_title IN VARCHAR2,
    p_img_desc IN VARCHAR2,
    p_categories IN VARCHAR2)IS


v_key INTEGER := NULL;
v_cat_list VARCHAR2(4000) := '';

BEGIN
 -- get next primary key and assign to variable as it is
  needed for foreign keys
 SELECT seq_img_id.nextval INTO v_key
 FROM dual;

/* p_id This is an IN OUT parameter and therefore, will be
returned to the calling application.*/
 p_id := v_key;

 -- insert into image_table (the master table)
 INSERT INTO image_table(img_id,
      img_title,
      img_desc)
VALUES
 (v_key,
 p_img_title,
 p_img_desc);

-- p_categories is a comma delimited list of image cate-
gories
v_cat_list := p_categories;

WHILE (LENGTH (v_cat_list) > 0 ) LOOP
 IF (INSTR(v_cat_list, ',') > 0) THEN
  v_cat := SUBSTR(v_cat_list, 0 , INSTR(v_cat_list, ',') -1 );

  v_cat_list := SUBSTR(v_cat_list, INSTR(v_cat_list, ',') +1 );
 ELSE
  v_cat := v_cat_list;
   v_cat_list := '';
 END IF;

  -- insert categories and foreign key link to the image
  table
  INSERT INTO img_category(category_id,
                           category_name,
                           img_id)
  VALUES (seq_category.nextval,
    v_cat_name,
    v_key);
     END LOOP;

 -- finally if there are no errors commit;
 COMMIT;
EXCEPTION

 WHEN others THEN
  /* when others is a
    default exception type that will catch all types of
   exceptions.*/
ROLLBACK;

END;
```

CODE LISTING

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

CF Community

Tales from the **List**

Access Isn't All Bad

BY **SIMON HORWITH**

The thread I've chosen to examine this month is all about driving ColdFusion sites with Microsoft Access databases. Very often I hear developers state that "Access is evil," or that you'd have to be crazy to use Access on the back end of a Web application, but are these generalized statements warranted? The **CFDJList** discussion began with a post by **Chelsey Gourgaud**, who wrote asking whether you can use Access when building a ColdFusion site, whether it will fail under stress, what other implications this practice carries, and what alternatives are available and commonly implemented. I must admit that I was quite surprised by the usage load that Access is successfully handling when driving CF sites, based on what developers described in their replies to this post.

Will Swain replied to the post, reassuring Chelsey that he uses Access 2000 for a lot of sites, including one that receives around 100,000 visitors per week. The skeptical side of me immediately wondered whether that number is accurate as well as what percentage of these visits resulted in data being inserted into the database or any performance-intensive query being run. Rather than immediately ask Will about this, I decided to wait and see what other list subscribers had to say about their experiences with Access. I found out that Will is not alone.

Ray Thompson, a very active and knowledgeable subscriber, chimed in and stated that he has a site that queries indexed columns in a 500,000-record Access database table in about 20 milliseconds! I have to assume that the query is rather simple, and that it doesn't perform any "real" text searching (just AutoNumber and/or number fields), but this is still not only surprising, but impressive.

Ray did go on to list some of the limitations of Access. Specifically, it doesn't have support for stored procedures, has a one-gigabyte size limitation, doesn't have support for concurrent users, and has to be "known to the ColdFusion Server." Meaning that unlike enterprise databases, which can sit behind a firewall and "listen" for requests, Access has to sit in front of the firewall on the machine running ColdFusion. He warned of this security risk, and recommended setting the number of connections ColdFusion uses to connect to the Access database to two (in the ColdFusion Administrator) in order to deal with the concurrency issues.

Other members (myself included) warned Chelsey of the concurrency issues surrounding Access – that it was never intended for use by simultaneous users. **Douglas Knudsen** added to the discussion by recommending that Chelsey visit the Macromedia CFQUERY FAQ at www.macromedia.com/v1/handlers/index.cfm?ID=22128&Method=Full.

A number of List members posted regarding their preferences and experiences with other databases. Many recommended going to SQL Server, or to MySQL if SQL Server isn't within the budget; some also recommended Oracle. As it turns out, Chelsey downloaded and installed MySQL to give it a try, which also sparked a thread from people interested in helping Chelsey properly configure MySQL datasources for use with ColdFusion.

• • •

All in all, we learned quite a bit about Access and about how ColdFusion handles database connections from this thread. I, for one, learned a lot about its popularity among ColdFusion developers. While I still advise running any Web site off an enterprise-level database, I can now state that there are people successfully running real-world applications off Access databases, and that Access may be a good short-term/low-budget database solution. Think about this next time, before you immediately begin preaching "the evils of Access," and join us next month for yet another installment of **Tales from the List**.

@ 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).*

CFDJnews

Free CFMX Development Accounts

(Newark, DE) – HostMySite.com is offering a free trial of Web hosting accounts with Macromedia ColdFusion MX support. The current version running is RC final 48097.

The free accounts include 200MB of Web space and a 100MB SQL Server 2000 database to be used as part of the testing process.

HostMySite.com is also offering free setup to anyone who converts their free testing account to a paid CFMX hosting account when the release version is available (there's no obligation to do so, however). The duration of the trial period is limited (the current release candidate expires once the commercial version is available), and the testing accounts are strictly for testing and nonproduction use. Requests for a trial account can be placed at www.HostMySite.com/cfmx.hms

PaperThin Updates CommonSpot Content Server

(Boston) – PaperThin, Inc., has released CommonSpot Content Server 3.0, a full-featured, browser-based Web publishing and dynamic content management solution.

New features include an improved administrator interface, integrated link management tools, and expanded scheduling and reporting options. CommonSpot Content Server was named Best Content Management Tool by the 2001 *CFDJ* Readers' Choice Awards. www.paperthin.com

Special Offer for ColdFusion MX Users

(San Francisco) – For a limited time, Macromedia ColdFusion MX users can save when they subscribe to ***ColdFusion Developer's Journal***, the only magazine devoted exclusively to ColdFusion topics.

CFDJ delivers up-to-the-minute information every month directly to your mailbox. This includes server-side programming tips and tricks, database programming techniques, a monthly how-to column by Ben Forta, product reviews, CF usergroup information, Q&A with ColdFusion experts, and full working source code samples.

The special subscription prices below include a FREE monthly ***CFDJ*** Digital Edition.

Two years/24 issues: $148 U.S. ($67.76 off newsstand price)
One year/12 issues: $79.99 U.S. ($27.89 off newsstand price)

According to Jeremy Allaire, CTO of Macromedia, "***CFDJ*** is a great resource for the ColdFusion developer community and demonstrates the incredible influence of CFML on the Web development universe."

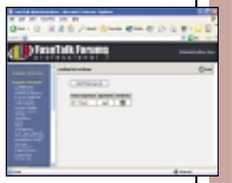
To take advantage of this limited special offer, go to www.macromedia.com/software/coldfusion/special/cfdj/.

EVENT

Macromedia DevCon 2002
October 27–30
Lake Buena Vista, FL
www.macromedia.com/v1/conference/

Fusetalk E-Mail List Synch Module from e-Zone Media

(Ottawa, ON) – e-Zone Media has released the FuseTalk E-Mail List Synch Module. The module allows organizations using FuseTalk, a discussion forum solution for ColdFusion environments, to integrate and synchronize e-mails and e-mail recipient lists with discussion forums to more closely link the two user communities.

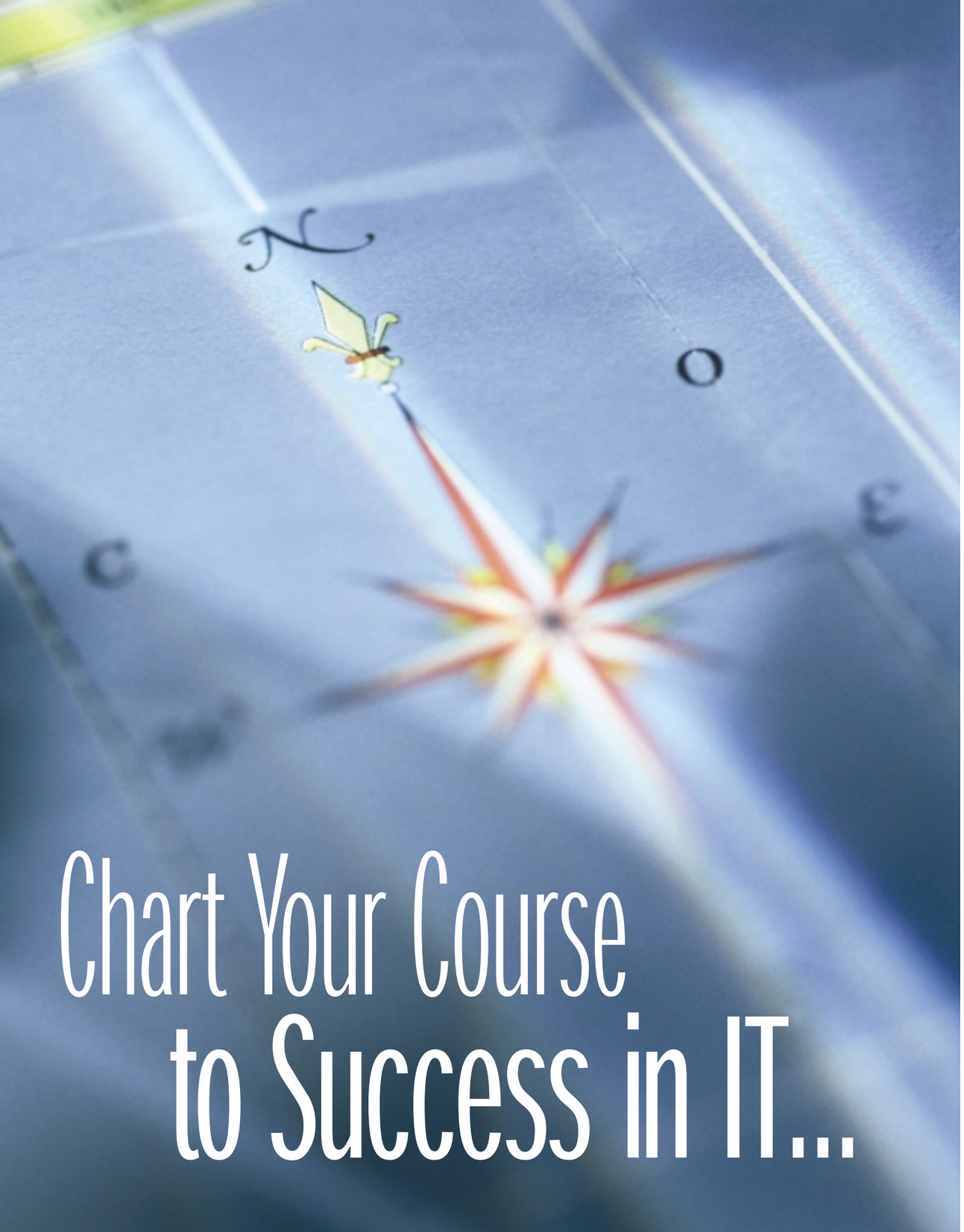


The module automatically posts e-mails sent to e-mail subscribers on the forums so that forum users can receive the same information. Likewise, administrators can have selected posts or threads automatically turned into an e-mail that can be sent to the e-mail subscriber community. A range of options gives forum and list administrators flexibility and control.

The module is available in both professional and enterprise versions. www.e-zonemedia.com

CFDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
ACTIVE PDF	WWW.ACTIVEPDF.COM	949.582.9002	4
CFADVISOR	WWW.CFADVISOR.COM		47
CFDYNAMICS	WWW.CFDYNAMICS.COM	800.422.7957	15
CFXHOSTING	WWW.CFXHOSTING.COM	866.CFX.HOST	39
CTIA	WWW.CTIASHOW.COM		9
ENGINDATA RESEARCH	WWW.ENGINDATA.COM	201.802.3082	50,51
E-ZONE MEDIA	WWW.FUSETALK.COM	866.477.7542	33
HOSTMYSITE.COM	WWW.HOSTMYSITE.COM	877.215.HOST	31
INTERMEDIA	WWW.INTERMEDIA.NET	800.379.7729	52
JDJ STORE	WWW.JDJSTORE.COM	888.303.JAVA	45
MACROMEDIA	WWW.MACROMEDIA.COM/GO/CFMXAD	877.460.8679	11
MACROMEDIA	WWW.MACROMEDIA.COM/GO/USERGROUPS	877.460.8679	13
NEW ATLANTA	WWW.NEWATLANTA.COM		3
PACIFIC ONLINE	WWW.PACONLINE.NET	877.503.9870	25
PAPERTHIN	WWW.PAPERTHIN.COM	800.940.3087	23
RACKSHACK	WWW.RACKSHACK.NET	800.504.SURF	2
SYS-CON MEDIA	WWW.SYS-CON.COM/SUBOFFER.CFM	201.802.3069	41
WEB SERVICES EDGE WEST	WWW.SYS-CON.COM	201.802.3069	21
WEB SERVICES EDGE	WWW.SYS-CON.COM	201.802.3069	35
WEB SERVICES JOURNAL	WWW.SYS-CON.COM	201.802.3069	43

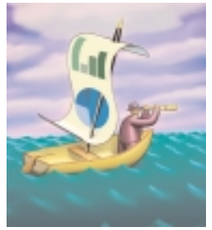


...order your copy of

Java Trends: 2003

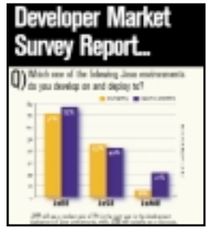
Available July 1*

Don't go astray. In the vast sea of Internet technology, market conditions change constantly. Will Java remain the hot platform it is today? Will C# rapidly gain ground? What are the world's foremost Java developers aiming their sights toward? Which companies come to mind when planning their next project? How can their thinking direct your efforts?



EnginData Research has studied the IT industry extensively, spotting many key trends and alerting industry leaders along the way. In the first quarter of 2002, they embarked on their most challenging mission ever: the most in-depth study ever done of the Java development marketplace.

After months of analyzing and cross-referencing more than 10,500 comprehensive questionnaires, the results are now available.



Here's just a sample of the critical data points the Java report delivers...

- ✔ Current and projected usage of languages and platforms
- ✔ Multiple rankings of hundreds of software vendors and tools
- ✔ Types of applications being developed
- ✔ Databases being used
- ✔ Purchasing patterns
- ✔ Company size
- ✔ Development and purchasing timelines
- ✔ Perceptions and usage of Web services
- ✔ Preferred Java IDE
- ✔ J2EE, J2ME, J2SE usage comparisons

As an IT specialist, **EnginData Research** focuses exclusively on three leading drivers in IT today – Java, Web services, and wireless development. Coming soon, our Web services survey will deliver such critical knowledge as:

- ✔ Time frame for developing Web services – enabled apps
- ✔ Percentage of apps with Web services today
- ✔ Sourcing and hosting Web services
- ✔ Perceived leaders in Web services tools and solutions

EnginData's research is invaluable to leading IT vendors, integrators, Fortune 500 companies, trade-show organizers, publishers, and e-business organizations worldwide.

Navigate your company through the challenging IT marketplace with the trusted knowledge and intelligence of **EnginData Research**. Order your copy of the 2002–2003 Java market study by calling Margie Downs at 201-802-3082, or visiting our Web site.

engindata RESEARCH

www.engindata.com

*A limited number of preview copies will be available at our booth (#624) at JDJEdge International Java Developer's Conference & Expo, June 24–27, at the Jacob Javits Convention Center in New York.