AN ADVANCE LOOK AT BLACKSTONE

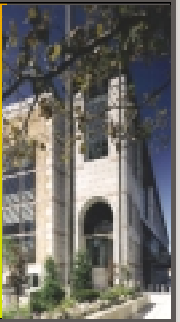
ColdFusion Developer's Journal

ColdFusionJournal.com

July 2004 Volume: 6 Issue: 7

Leveraging JDBC or Just Fetching Coffee?

By Isaac Dealey page 8



web services EDGE conference&expo

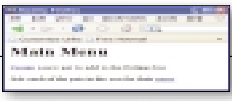
Web Services Edge 2005 East

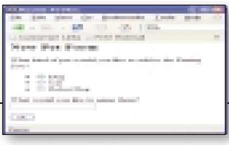
February 15-17, 2005

Hynes Convention Center

Boston, MA

See page 37 for details





RETAILERS PLEASE DISPLAY UNTIL SEPTEMBER 30, 2004

$9.99US $9.99CAN

0 09281 02689 1 08

SYS-CON MEDIA

edgemodern.com

“Macromedia® Dreamweaver® MX 2004 has helped us design and build a fully functional site which helps us immediately respond to customer needs–all without a Ph.D. in web development.”

Drew Sanocki, EDGE*MODERN, Co-founder.

gulfstream.com

“With Dreamweaver, we’ll be able to take even greater advantage of CSS, which will add huge efficiencies to how we develop and maintain our website.”

Will Dent, Gulfstream Aerospace, Interactive Marketing.

Dreamweaver. The highest common denominator.

Now updated and
up to 70% faster.

See more at: macromedia.com/go/dwupdated

It’s everybody’s PDF™

Finally, a software company that offers affordabe yet flexible PDF solutions to meet every customer's needs. Using activePDF™ to automate the PDF creation process eliminates the need for end-user intervention so your employees can concentrate on what they do best.

Licensed per server, activePDF solutions include a COM interface for easy integration with ColdFusion. Dynamically populate PDF forms with information from a database, convert ColdFusion web pages to PDF on the fly, dynamically print reports to PDF using CF and much more. Users can also merge, stitch, stamp, secure and linearize PDF, all at a fraction of the cost of comparable solutions. Download your free trial version today!

activePDF®
Leading the iPaper Revolution

www.activePDF.com

editorial

editor-in-chief
Robert Diamond robert@sys-con.com
technical editors
Raymond Camden raymond@sys-con.com
Simon Horwith simon@sys-con.com
executive editor
Jamie Matusow jamie@sys-con.com
editor
Nancy Valentine nancy@sys-con.com
associate editors
Gail Schultz gail@sys-con.com
Jean Cassidy jean@sys-con.com
Jennifer Van Winckel jennifer@sys-con.com
research editor
Bahadir Karuv, PhD bahadir@sys-con.com

production

production consultant
Jim Morgan jim@sys-con.com
lead designer
Abraham Addo abraham@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
Tami Beatty tami@sys-con.com

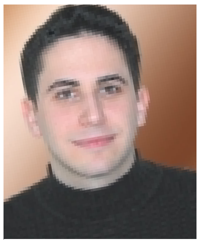
contributors to this issue

Isaac Dealey, Ben Forta, Hal Helms, Simon Horwith, Jeffry Houser, Nate Nelson, Grant Szabo

editorial offices

SYS-CON MEDIA
135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 **FAX:** 201 782-9638
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

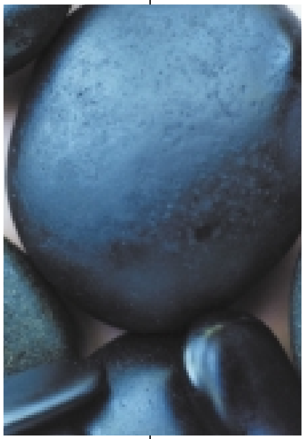
Blackstone...



By Robert Diamond

The name on everyone's lips these days is *Blackstone*, the current code name for the next release of ColdFusion, which is coming in early 2005. Look for beta testing to start later in 2004, with info on Macromedia's site when that becomes available.

Folks around the country have been getting a sneak peek at many of the features from Ben Forta, who's been on a whirlwind user group tour that just wrapped up. How he travels so much, I don't know, but thankfully for those of us bound to the homefront, he's put a report online at www.macromedia.com/devnet/mx/coldfusion/articles/blackstone.html. Check it out for details on some of the new features of Blackstone, along with some general updates on the "State of the CF Union."



I couldn't let this month's editorial go by without sharing a few of these exciting highlights. For me, the most interesting new features include improvements to data entry, because these have always been among the most time-consuming parts of development. Forms are generally a nightmare, plain and simple, and even when you're at a point when for the most part you're reusing existing code, they're still a pain. The new set of CFForm and related tags will generate XML XForms that can then be skinned using XSL, something that'll greatly open up form reuse across applications, simplifying many of these common tasks. Saying that I can't wait for that would be an understatement.

Another addition is one I've been hoping for for awhile now: Flash-based versions of some of those old Java tags like CFTree and CFGrid, which were long overdue for a good overhaul. These have always been useful on the admin end, and a new interface should make them both useful and usable.

Also included in Blackstone, and worth highlighting, are some new features of printing and reporting, two of the things that ColdFusion has always needed a bit more help with. There's a new tag, CFDocument, that you can wrap around content to render printable versions of it. It has a slew of options, including PDF and Macromedia's new FlashPaper technology as output types. For reporting, the CFReport tag has been given an update, with a new report type known as a "CFR." A CFR is an XML-defined, banded report with everything you could want, making it much easier to manipulate and use.

There are tons of other features also, and definitely "something for everyone," so I urge you to go take a look at the original DevNet article to learn more about them and which ones will apply the most to you. All developers are of course different, and this release, probably more so than many in the past (as the underlying Java platform has stabilized), is a direct result of customer requests and feedback. Overall, Blackstone helps to reaffirm Macromedia's continuing commitment to the world of ColdFusion development. They're in it for the long run, and as CF nears its 10th birthday, that's a very good sign. The best – as they say – is yet to come!

About the Author

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

CommonSpot™.

Efficient Content Management

fast. easy. affordable.

- 100% browser-based
- Content object architecture
- Template driven pages
- 50+ standard elements
- Extensible via ColdFusion
- Content reuse
- Content scheduling
- Flexible workflow
- Granular security
- CSS support
- 508 compliance
- Personalization
- Replication
- Custom metadata
- Custom authentication
- Static site generation
- Multilanguage support

With CommonSpot Content Server you get it all. CommonSpot's exceptional blend of rapid deployment, ease of use, customization and scalability make it the leading ColdFusion content management solution.

Our rich Web publishing framework empowers developers with out-of-the-box features such as template-driven pages, custom content objects, granular security, flexible workflow and powerful ColdFusion integration hooks (just to name a few), allowing you to rapidly build and efficiently deploy dynamic, high performance Web sites.

For larger implementations, CommonSpot scales efficiently, delivering enterprise-level capabilities like replication, static content generation, multi-site clustering, personalization and custom authentication, across a diverse set of platforms.

For the past six years, PaperThin has been a leader in the ColdFusion community, and CommonSpot has been the solution of choice for organizations of all sizes, including AFL-CIO, Boeing, Kaiser Family Foundation, Ohio University, PGA.com and hundreds of others. CommonSpot's sophisticated feature set and affordable pricing are an unbeatable combination.

Call us today at **800.940.3087** to schedule a live demonstration of your site running under CommonSpot, or visit **www.paperthin.com** to learn more.

Paper | Thin

SYS-CON MEDIA

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

vp, business development
Grisha Davida grisha@sys-con.com

group publisher
Jeremy Geelan jeremy@sys-con.com

advertising

senior vp, sales & marketing
Carmen Gonzalez carmen@sys-con.com

vp, sales & marketing
Miles Silverman miles@sys-con.com

advertising director
Robyn Forma robyn@sys-con.com

director of sales & marketing
Megan Mussa megan@sys-con.com

associate sales managers
Kristin Kuhnle kristin@sys-con.com
Beth Jones beth@sys-con.com
Dorothy Gil dorothy@sys-con.com

sys-con events

president, events
Grisha Davida grisha@sys-con.com

conference manager
Lin Goetz lin@sys-con.com

customer relations

circulation service coordinators
Edna Earle Russell edna@sys-con.com
Linda Lipton linda@sys-con.com

manager, jdj store
Brunilda Staropoli bruni@sys-con.com

sys-con.com

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

web designers
Stephen Kilmurray stephen@sys-con.com
Matthew Pollotta matthew@sys-con.com

online editor
Lin Goetz lin@sys-con.com

accounting

financial analyst
Joan LaRose joan@sys-con.com

accounts payable
Betty White betty@sys-con.com

accounts receivable
Shannon Rymsza shannon@sys-con.com

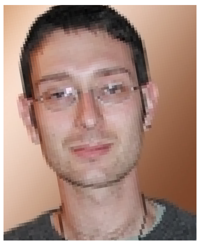
subscriptions

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

Tales from the List

Dynamic variables

This month, rather than focusing on a more advanced or more obscure task, I'm going to talk about a task which I consider more fundamental – dynamic form variable names and their retrieval on the action page. It seems that not a month goes by without a post to the *CFDJ* List (or other kindred lists) inquiring about how to create and/or access form variables with dynamic names.



By Simon Horwith

The thread that prompted me to write about this was started by Mike from NZSolutions, who wrote to the List stating that he has a form with 10 checkboxes named "size1" to "size10". Each checkbox has a unique value and Mike is curious about how to use the checkboxes the user selects on his action page without having to write 10 "<CFIF isDefined()..." statements.

Certainly there are ways to do this, but the first rule of development is never to do more work than is necessary. George Farnsworth immediately responded, noting that Mike could give every checkbox the same name but a different value. Then on the action page Mike would only have to detect the existence of that one variable or use <CFPARAM> to ensure its existence. The form variable value would then be a comma-delimited list of the values of each of the selected checkboxes from the form.

Clearly, this is the best approach if it suits Mike's needs. But what if the checkboxes must have unique names? Dave Deeds posted to the thread and mentioned that there's a reserve variable, "form.fieldnames", which contains a comma-delimited list of the names of all form fields that were posted to the server. Because form scope variables are created only for those checkboxes that were "checked" when the form was submitted, only the names of those checkboxes would exist in this list. I chimed in and mentioned that although it's a tiny bit more work, I prefer to use the structKeyList() function since it keeps your syntax consistent. Not only will StructKeyList() return the names of all of the variables in the form scope, but the same syntax is used to retrieve the names of all variables in the URL, session, variables, or any other scope or structure.

Dave followed up the thread with the kind of post I always love to see. Not only has he gotten the answer to his question but he has been exposed to a whole new realm of functionality he never knew about before. Dave's e-mail was to let the list know that his curiosity was piqued at the idea of treating "form" as just another structure, and so he was playing around with code to sort what's in the form scope and ways to loop over everything in the form scope and output name value pairs. His code for that looked something like:

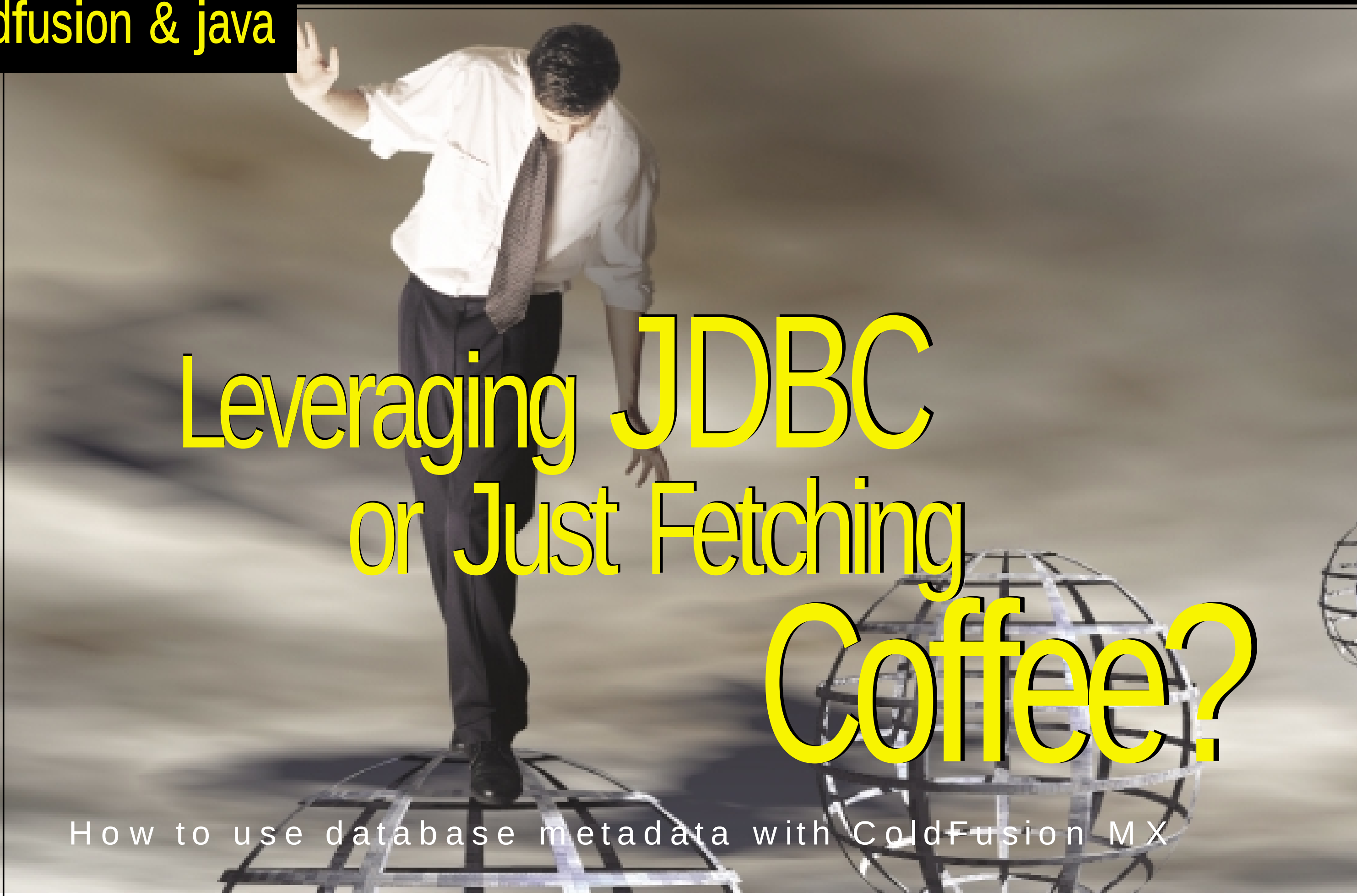
```
<cfoutput>
<cfloop collection = #form# item = "frm_field">
#frm_field# #StructFind(form, frm_field)#<br>
</cfloop>
</cfoutput>
```

He finished by stating that he "loves playing...and learning."

— continued on page 36

About the Author

Simon Horwith is co-technical editor of CFDJ, *and chief technology officer of eTRILOGY Ltd., a software development company based in London, England. Simon has been using ColdFusion since version 1.5 and is a member of Team Macromedia. He is a Macromedia Certified Advanced ColdFusion and Flash developer and is a Macromedia Certified Master Instructor. In addition to administering the* CFDJ *List mail list and presenting at CFUGs and conferences around the world, he has also been a contributing author of several books and technical papers.*
simon@horwith.com



Leveraging JDBC or Just Fetching Coffee?

How to use database metadata with ColdFusion MX

By Isaac Dealey

Using JDBC metadata, it is now possible to analyze databases and automate database tasks with ColdFusion MX in ways that were problematic or at best difficult in the past. Although Java can be rather intimidating, you don't need to be a Java expert (or a database expert) to use these tools.

"Metadata" is all the information about how your data is stored (such as table names, column names, data types and sizes) as opposed to the data itself (such as names of people and meeting or appointment dates). All databases must store this information to function and many databases expose metadata (usually via SQL), although with each database making its own decisions about how the data is exposed, writing database-agnostic tools using SQL could be an arduous, frustrating, and often fruitless task.

Prior to ColdFusion MX the popular means of connecting to databases involved the Open Database Connectivity (ODBC) standard, which used an API implemented in the operating system to connect applications to their databases for both executing SQL and retrieving metadata. While metadata is available through ODBC, it remains difficult to implement using C++ and typically via the recently deprecated COM standard. With the introduction of CFMX the supported means of connecting to databases became the Java Database Connectivity (JDBC) standard, which includes a similarly robust API for retrieving metadata that is much more accessible. (Connection to ODBC datasources is still supported, although it requires the use of a JDBC-ODBC bridge driver.) The additional abstraction provided by Java and the new JDBC standard allows us to use a common syntax for all our metadata tasks with any database supporting the standard and easily implemented in simple CFML tags or functions.

For example, you might create a function or a CFC method to fetch all the column names, data types, sizes, and nullability of columns in a specified database table. This is just one example of a number of JDBC-driven features I've implemented in the onTap Framework (described in "Features Without Fixtures" in *CFDJ*, Vol. 6, Issue 4), and is merely scratching the surface of the JDBC metadata potential.

Before You Leap

Before you make the decision to leverage JDBC metadata in your applications, there are a number of things to consider.

First, the technique described in this article requires use of the coldfusion.server.ServiceFactory object. You may recognize this object from some other essays, articles, and mailing lists. This is one of the key objects the ColdFusion server uses internally to manage its processes and has been the subject of much discussion since the release of ColdFusion MX. Because this class is not a documented feature of CFML or the ColdFusion server, Macromedia won't provide support for any CF software using it, and won't guarantee the availability of these features in the future. This is the primary concern with regard to using JDBC directly in your CF applications as described in this article, and may be the deciding factor in determining if you want to use these features in a specific project. Although these features are undocumented and not supported by Macromedia, it's important to note that JDBC is a well-defined standard and the ColdFusion Server Administrator application uses these same features for managing the server's DSNs. While this doesn't guarantee that these features will not change it is likely that the same functionality will be reproducible with any future versions of the ColdFusion Server.

Because ODBC includes no specifications for metadata, the techniques described in this article are not compatible with the ColdFusion Server's ODBC-Bridge drivers.

Unfortunately I also have some bad news for users of Microsoft Access databases. The MS Access database doesn't conform to the standards for JDBC, requiring the use of ODBC to make connections to Access even when the ODBC-Bridge drivers are not being used. Because the ODBC Bridge driver included with ColdFusion MX doesn't implement all the JDBC metadata features, the available features when using MS Access are particularly limited. I have been able to retrieve a list of supported table-types in an Access database, but not tables or columns. Metadata can be retrieved from Access by querying its hidden tables (which also provides rather limited information) or by implementing a separate COM solution – although this was recently deprecated by Microsoft so providing common syntax for this may be quite a challenge.

Finally because this technique requires creating a connection to your database directly through Java (bypassing the ColdFusion Server), you need to include the database name (in addition to the DSN), username, and password of your database in your application code. Generally speaking this should be a trivial consideration, although some developers may dislike it as a matter of style partly because some of this information must also be included in the ColdFusion Administrator, increasing the number of independent locations where the information is stored and must be updated when changed. Using this information in the application may be similar to using DSN-less database connections with ColdFusion 5 and a connection-string, although this feature is not implemented in CFMX.

Java and Databases: An Introduction

If you don't have much experience with Java and you're thinking this article might be a bit over your head, don't worry. Although Java can be rather intimidating, you don't need to be a Java expert (or a database expert) to use these tools.

For the purpose of this article it's important that you understand the distinction between a Data Source Name (DSN), a Relational Database Management System (RDBMS), and a catalog or database. An RDBMS is a database server such as SQL Server, Oracle, Sybase or PostgreSQL. Each RDBMS installation may (and almost always does) contain multiple individual databases. In RDBMS parlance individual databases are often known as "catalogs" (called "databases" in SQL Server, and "schemas" in Oracle). By contrast the DSN is the name given to a connection pointing to an individual database or catalog (within a specific RDBMS installation) in the ColdFusion Server Administrator or J2EE application server. A DSN is often the same as the name of the database or catalog within the RDBMS, although the two are not related and not required to match. It is also important to note that catalogs or databases also include what are known as "schemas." A schema is simply a codified collection of tables, views, stored procedures, and other items within an individual database. Usually an individual database will contain or use only one schema (for instance, with Microsoft SQL Server all data is usually stored in a "dbo" schema).

Beyond this database knowledge, these terms may also be helpful in understanding the Java involved in this process:

- ***OO or OOP (Object-Oriented Programming):*** This article will not focus on OOP at large (a subject on which many entire books have been written). All you need to know is that OOP is the philosophy of Java and the origin of the terms Class, Object, Method, and Interface.
- ***Class:*** A class is a blueprint (or template) for creating objects. An object is a complex data type containing both variables (also called "properties") and functions (also called "methods"). Class functionality in ColdFusion is provided by ColdFusion Components (CFCs).
- ***Object:*** An object (sometimes called an "instance") is a variable created using a class as its blueprint. Thus all objects of a specific class have the same properties (variables) and methods (functions) although the properties of individual objects or instances may have different values. Objects are created in CFML using the cfobject tag or the CreateObject function.
- ***Method:*** A method is a function within a class and its objects. Methods in CFCs are often referred to as "CFC functions".
- ***Interface:*** A public definition of properties and methods required by certain classes. In simple terms an interface contains a list of properties and methods that certain classes must contain. A more in-depth understanding of interfaces is not necessary for this article.

Getting Connected, Getting Results

Although CFML included simple tools for executing SQL queries from its earliest days, connecting to a database to do anything other than manipulate data has never been addressed. Simplified native tools to do this in CFML (like the cfquery tag) do not yet and may never exist, so arguably the best solution for retrieving database metadata lies in Java. To begin our journey into the land of JDBC we must start by creating a Java object (a process sometimes referred to as "instantiating a class"). Create a new ColdFusion template called jdbc.cfm. Place the following lines of code in this new template and execute it:

```
<cfscript>
dss = CreateObject("java",
  "coldfusion.server.ServiceFactory");
dss = dss.getDataSourceService();
</cfscript>
<cfdump var="#dss#">
```

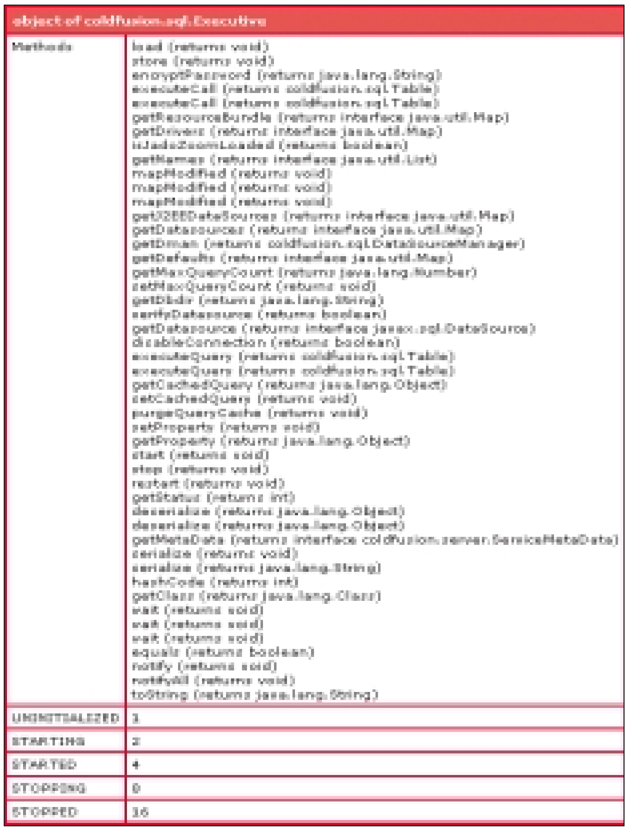
The output of the cfdump tag should look like Figure 1. This is how the ColdFusion server's cfdump tag displays a Java object. At the top you'll see "object of coldfusion.sql.Executive". This tells you that the variable dumped (dss) is an object and that the object uses the coldfusion.sql.Executive class as its blueprint. To retrieve metadata from your database, you'll need to use the dss object you've just created to create a metadata object by adding this code to your jdbc.cfm template:

```
<cfscript>
conn = dss.getDataSource("yourDSN");
conn = conn.getConnection("username","password");
mdata = conn.getMetaData();
</cfscript>
<cfdump var="#mdata#">
```

In this code, yourDSN is the name of your data source as specified in the ColdFusion Server Administrator. Username and password are the username and password required to access the database. Although you can omit username and password when executing queries using cfquery (when the username and password are stored in the ColdFusion Server Administrator DSN definition), you can't here. The username and password of your database must be used in the getConnection() method, otherwise your Java object will not connect to your database. The mdata variable you've created is an Object that now displays a class name of macromedia.jdbc.base.BaseDatabaseMetaData.

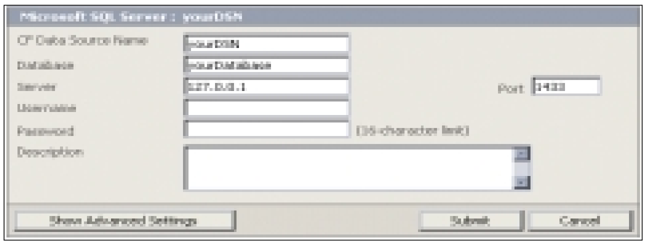
At this point you might want to open a new browser and view the documentation for JDBC on the Sun Microsystems Web site: http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html

At the top of this page, you should see the words "Package



object of coldfusion.sql.Executive	
Methods	load (returns void) store (returns void) encryptPassword (returns java.lang.String) executeCall (returns coldfusion.sql.Table) executeCall (returns coldfusion.sql.Table) getResourceBundle (returns interface java.util.Map) getDrivers (returns interface java.util.Map) isJadoZoomLoaded (returns boolean) getNames (returns interface java.util.List) mapModified (returns void) mapModified (returns void) mapModified (returns void) getJ2EEDataSources (returns interface java.util.Map) getDatasources (returns interface java.util.Map) getDman (returns coldfusion.sql.DatasourceManager) getDefaults (returns interface java.util.Map) getMaxQueryCount (returns java.lang.Number) setMaxQueryCount (returns void) getDbdir (returns java.lang.String) verifyDatasource (returns boolean) getDatasource (returns interface javax.sql.DataSource) disableConnection (returns boolean) executeQuery (returns coldfusion.sql.Table) executeQuery (returns coldfusion.sql.Table) getCachedQuery (returns java.lang.Object) setCachedQuery (returns void) purgeQueryCache (returns void) setProperty (returns void) getProperty (returns java.lang.Object) start (returns void) stop (returns void) restart (returns void) getStatus (returns int) deserialize (returns java.lang.Object) deserialize (returns java.lang.Object) getMetaData (returns interface coldfusion.server.ServiceMetaData) serialize (returns void) serialize (returns java.lang.String) hashCode (returns int) getClass (returns java.lang.Class) wait (returns void) wait (returns void) wait (returns void) equals (returns boolean) notify (returns void) notifyAll (returns void) toString (returns java.lang.String)
UNINITIALIZED	1
STARTING	2
STARTED	4
STOPPING	8
STOPPED	16

Figure 1:Object of macromedia.jdbc.base.BaseResultSet



Microsoft SQL Server : yourDSN			
CF Data Source Name	yourDSN		
Database	yourDatabase		
Server	127.0.0.1	Port	1433
Username			
Password		(16-character limit)	
Description			

Figure 2: ColdFusion Administrator DSN configuration

java.sql". Don't worry; you won't need to learn everything there is to know about Java or JDBC. Suffice to say that everything that is JDBC can be found in this java.sql package. The term package is merely used to describe a collection of related classes and interfaces. Below this title you should see a table labeled "Interface Summary." In the interface summary you'll notice an interface named DatabaseMetaData. Because the mdata object is a DatabaseMetaData object, any properties and methods found in the DatabaseMetaData interface will also be available in the mdata object.

When you select the DatabaseMetaData link you'll see a page with some general information about the interface, followed by a "Field Summary" table (these are the interface properties) and a "Method Summary" table (these are the interface functions). In the Method Summary you'll notice a number of methods beginning with the word "get," such as getCatalogs and getColumns. These "getter" methods are often referred to as "accessors" and are used to fetch information from the object. In this case these accessor methods are used to fetch metadata information from the RDBMS.

Have a look at the getCatalogs method (see also Figure 3). You'll notice that there is a brief description of the method beneath the name, and the name is linked to a more thorough description of the method further down on the same page. To the left of the linked method name you'll notice a link labeled "ResultSet". The column to the left of the method names displays the type of data returned by each of the methods (with a link to documentation for the specified class or interface if applicable). The functions you will likely find most useful when working with JDBC will return ResultSet objects. A ResultSet is the Java equivalent of a ColdFusion query variable created by the cfquery tag or a number of other tags and functions such as cfdirectory and cfpop.

Returning to our jdbc.cfm template, let's add the following code and execute the template:

```
<cfset rsCatalog = mdata.getCatalogs()>
<cfdump var="#rsCatalog#">
```

You Call This Results?

I have stated that a ResultSet is the Java equivalent of a ColdFusion query variable and that the method mdata.getCatalogs() will return a Java ResultSet object. Something is happening here, however, because when you display the new rsCatalog variable (a ResultSet object) using the cfdump tag, it produces something quite different than what is displayed when you use the same tag to display a query. You might expect the display to resemble Figure 4, but instead it resembles Figure 1.

BlueDragon [6.1]

RELEASED!

RUN YOUR CFML:

Within BlueDragon Server, or as a native .NET or J2EE web application.

On the platform, web server, and operating system of your choice.

New Atlanta
COMMUNICATIONS

Get BlueDragon hosting:

CFDynamics is now offering BlueDragon hosting!
Find out more by visiting: www.cfdynamics.com/bluedragon

As one of the ColdFusion hosting community leaders, CFDynamics is constantly looking for ways to deliver new and cutting edge technologies to the ColdFusion community. We are excited to announce our premiere partnership with New Atlanta by offering BlueDragon hosting. We look forward to seeing how BlueDragon expands the possibilities of ColdFusion. Come join CFDynamics and New Atlanta in expanding the ColdFusion horizon!

POWERFUL HOSTING PLANS

- FREE SQL server access
- FREE account setup
- UNLIMITED email accounts
- GENEROUS disk space / data transfer
- 30 day MONEY-BACK GUARANTEE
- GREAT VALUE!

RELIABLE NETWORK

- 99.99% average uptime!
- State-of-the-art data center has complete redundancy in power, HVAC, fire suppression, bandwidth, and security
- 24 / 7 network monitoring

FANTASTIC SUPPORT SERVICES

- Comprehensive online support
- Knowledgeable phone support
- We focus on your individual needs

CFDynamics
A Division of Konnections Inc.

SIGN UP FOR A HOSTING ACCONT TODAY!

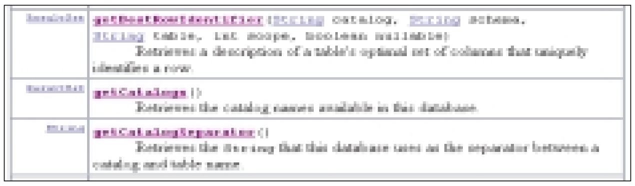
Use promo code: CFDJ04E and receive a FREE T-SHIRT!

macromedia® | ALLIANCE PARTNER

BlueDragonCertified
V.I.P. Hosting Partner

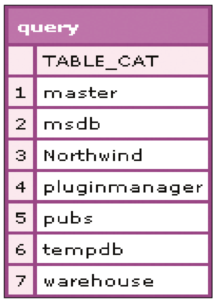
WWW.CFDYNAMICS.COM

866 - CFDYNAMICS
866-233-962-6427



ResultSet	getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable) Retrieves a description of a table's optimal set of columns that uniquely identifies a row.
ResultSet	getCatalogs() Retrieves the catalog names available in this database.
String	getCatalogSeparator() Retrieves the String that this database uses as the separator between a catalog and table name.

Figure 3: JDBC documentation on www.sun.com



query

	TABLE_CAT
1	master
2	msdb
3	Northwind
4	pluginmanager
5	pubs
6	tempdb
7	warehouse

Figure 4: Tables in your database

One of the great advantages of CFML is it simplicity. CFML masks much of the complexity of Java to make our work easier. To this end the ColdFusion Server uses another proprietary class (coldfusion.sql.QueryTable) to provide the much simpler cfoutput, cfloop, and cfdump functionality we all know and love. This functionality is not included in Java and therefore isn't a part of the ResultSet object. Thus you need to manually convert the ResultSet object into a query. To make this easier I've created a function to perform this task. The ColdFusion code is shown in Listing 1.

Place this function at the top of your jdbc.cfm template, then add this code to the bottom of your template:

```
<cfset qCatalog = rstoquery(rscatalog)>
<cfdump var="#qCatalog#">
```

The output of this new variable should resemble Figure 4 and might be much more like what you expected to see from the previous cfdump tag output. With this function we can fetch queries from our metadata object using any of the ResultSet methods of the java.sql.DatabaseMetaData interface.

Returning to the documentation for the DatabaseMetaData interface at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html we see several other useful methods such as getColumns(), getTables(), getPrimaryKeys(), getKeysImported(), getKeysExported(), getProcedures() and getProcedureColumns(). Other methods such as getSuperTables() and getTablePrivileges() might be useful only in certain specialized applications. Let's test our rsToQuery() function with the getColumns() method. Add this code to the bottom of your template:

```
<cfset rsColumns =
  mdata.getColumns("yourDB",
  "%","yourTable","%")>
<cfset qColumns = rstoquery(rscolumns)>
<cfdump var="#qColumns#">
```

As a reminder, "yourDB" in the code above should be the name of your database within your RDBMS, which may or may not be the same as your DSN within the ColdFusion Server Administrator (see Figure 2). The variable yourDB specifies the catalog (or database) from which you want to retrieve columns. The remaining three arguments of this method are schemaPattern, tableNamePattern and columnNamePattern. At this point you might notice some similarity between a SQL query and the getColumns function. The percent (%) symbol is used in both cases as a wild-card character. This provides very flexible and granular control over what data is returned by this function. You'll find that any of the arguments of the DatabaseMetaData methods ending with the word "pattern" behave in this manner, providing the same functionality for getTables() or getProcedures(). For now don't worry about the other methods; simply choose the name of a table you know is in your database to place in the tableNamePattern argument in place of the string "yourTable".

The cfdump output from this new qColumns variable shows some information you might find quite useful in developing applications, although these results are subtly deceptive. You might notice that several of the columns appear to be duplicated in some manner, such as the is_nullable and nullable columns. While these two columns can likely be used interchangeably without worrying about the results, the column data-type information requires further scrutiny.

The new query contains a "type_name" column containing the data types of the columns in your table. There are also three additional columns, "data_type", "source_data_type" and "sql_data_type" two of which appear to be numeric values, while "source_data_type" contain only empty strings with most databases. Unfortunately none of these four values will be particularly useful when attempting to automate something like the implementation of cfqueryparam tags. This is what the Java documentation has to say about these columns:

- DATA_TYPE int => SQL type from java.sql.Types
- TYPE_NAME String => Data source dependent type name, for a UDT the type name is fully qualified
- SQL_DATA_TYPE int => unused
- SOURCE_DATA_TYPE short => source type of a distinct type or user-generated Ref type, SQL type from java.sql.Types (null if DATA_TYPE isn't DISTINCT or user-generated REF)

From this documentation we can guess that the sql_data_type column was used in a previous version of JDBC but has since been deprecated. The source_data_type column is closely related to user-defined data_types and as such will not be populated unless user-defined types (UDTs) are included in the schema. Most databases don't contain any UDTs, resulting in empty string values. The type_name column is also problematic because it is data-source dependent, meaning the values in this column will vary from one RDBMS to the next. This includes values such as "smalldatetime" and "int identity" for MS SQL Server and possibly Sybase, while these values are meaningless in an Oracle or PostgreSQL database or within the standards for JDBC, ODBC, or even SQL.

This leaves us with the data_type column, which appears to be an integer. Obviously an integer value of 12 or 93 is also unhelpful in automating something like the cfqueryparam tag, so we need to translate these integer values to a useful string value. The solution to this problem isn't very intuitive, although it is described in the documentation for the java.sql.Types interface. I've created another function to make this process easier. You can see the code for that in Listing 2.

Now that you have this function, you can use it to convert the values in your data_type column from integers to string

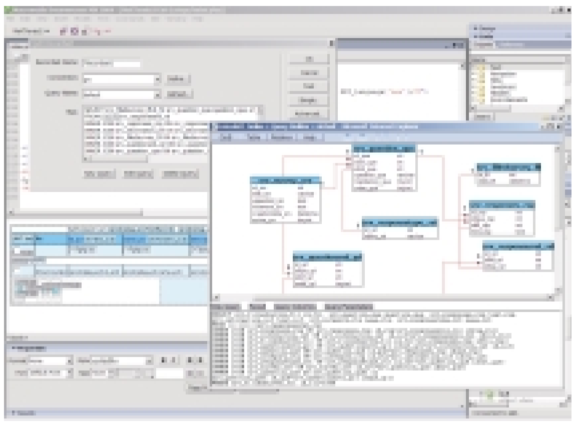
Dreamweaver Website Development

MANY needs - ONE solution

MX Kollection
Professional Dreamweaver Extensions
macromedia DREAMWEAVER 2004

MX Kollection

for ColdFusion and PHP



Create powerful dynamic lists

- Sort, filter and page result sets
- Perform multiple deletes
- Easily create Master/Detail lists

Build unified add/modify forms

- Client side and server side validation
- Insert into two tables
- Create form actions such as send mail or delete related record
- File/Image upload and resize

Create recordsets visually

- Build complex queries across multiple tables quickly

... and many more

The MX Kollection is a **suite of extensions** designed to **change the way you create dynamic web applications** with Dreamweaver MX.

ColdFusion and **PHP** developers will find our product enabling them to visually develop **e-Commerce, CMS, CRM, Backend** and other web solutions with increased efficiency, quality and capability.

Our customers think of it as **the next level in Dreamweaver MX visual software development.**

Download the demo and see the features and benefits of our extensions:
http://www.interaktonline.com/

MX Kollection - Professional web tools

Interakt
macromedia ALLIANCE PARTNER

values matching the appropriate SQL standard data-type names. These string values will make more sense and likely be more useful in your development. Add the jdbcType() function from Listing 2 to the top of your jdbc.cfm template and then add the following code to the bottom:

```
<cfloop query="qColumns">
  <cfset qColumns.data_type = jdbcType(qColumns.data_type)>
</cfloop>
<cfdump var="#qColumns#">
```

When you execute this template again, the results of the final cfdump tag output should display most of the information you might want about your table's columns in a friendly format you can use for your own development.

Comparing Notes

When working with the DataBaseMetaData object, be careful not to omit any of the arguments of its methods. Unlike many CFML functions, none of the arguments for these methods are optional. If you specify a method with the wrong number of arguments or with the wrong type of arguments (passing an array instead of a string for instance), the ColdFusion server will produce an error such as "The selected method [nameOfMethod] was not found." This can be confusing to those of us not familiar with Java when the documentation insists that the object does have a method with the specified name, and even more so when other scripts successfully implement the same method.

Concerning the data type of method arguments: when the Java documentation includes a pair of brackets [] to the right of an argument's data type, the expected argument should be an array of items of the specified type. For instance, "String[]" must be an array of strings and "int[]" must be an array of integers. This should not be an issue with most of the java.sql methods although the DatabaseMetaData.getTables() method's last argument is described in the documentation as "String[] types" (where "types" is the name of the argument). This can be easily accommodated by using something like listToArray("TABLE,VIEW") in your call to the getTables() method.

Finally, because the metadata in your database design isn't likely to change very often and because the ResultSet objects returned by the DatabaseMetaData methods must be converted to queries via a processor-intensive manual routine, it's a good idea to use the ColdFusion Server's persistent scopes (I recommend the server or application scope) to store the metadata for your database. To reduce server load in production, any metadata transaction involving conversion of a Java ResultSet object to a query should be given a consistent variable naming convention and retrieved from this cache instead of using JDBC if the desired data exists in the cache. For example: column data for your "tMembers" table might be stored in the variable server.dba.yourDSN.yourDB.dbo.tMembers.columns. If the structure of your tMembers table changes, you can restart the ColdFusion Server service or delete the server.dba.yourDSN.yourDB.dbo.tMembers variable to force the application to renew the data the next time it is requested.

Summary

Unfortunately a more thorough examination of the possibilities of using JDBC metadata in CF applications is beyond the scope of this article. After reading this far, however, you should be able to use Java in CFML to connect to your database, retrieve metadata ResultSets, and convert them to queries. With this knowledge and the JDBC documentation at your disposal, finding ways to use these features should be easy. I've implemented a number of features not discussed in this article in the onTap framework (www.turnkey.to/ontap). These include but aren't limited to alternatives to the cfinsert and cfupdate tags, automated query filtering using form and URL variables, and an alternative to the cfstoredproc tag which makes stored procedures as easy to use as cfinsert and cfupdate. Please download the framework to experiment with these features. I look forward to seeing how others use JDBC.

About the Author

Isaac Dealey has worked with ColdFusion since 1997 (version 3) for clients from small businesses to large enterprises, including MCI and AT&T Wireless. He evangelizes ColdFusion as a volunteer member of Team Macromedia, is working toward becoming a technical instructor, and is available for speaking engagements.

info@turnkey.to

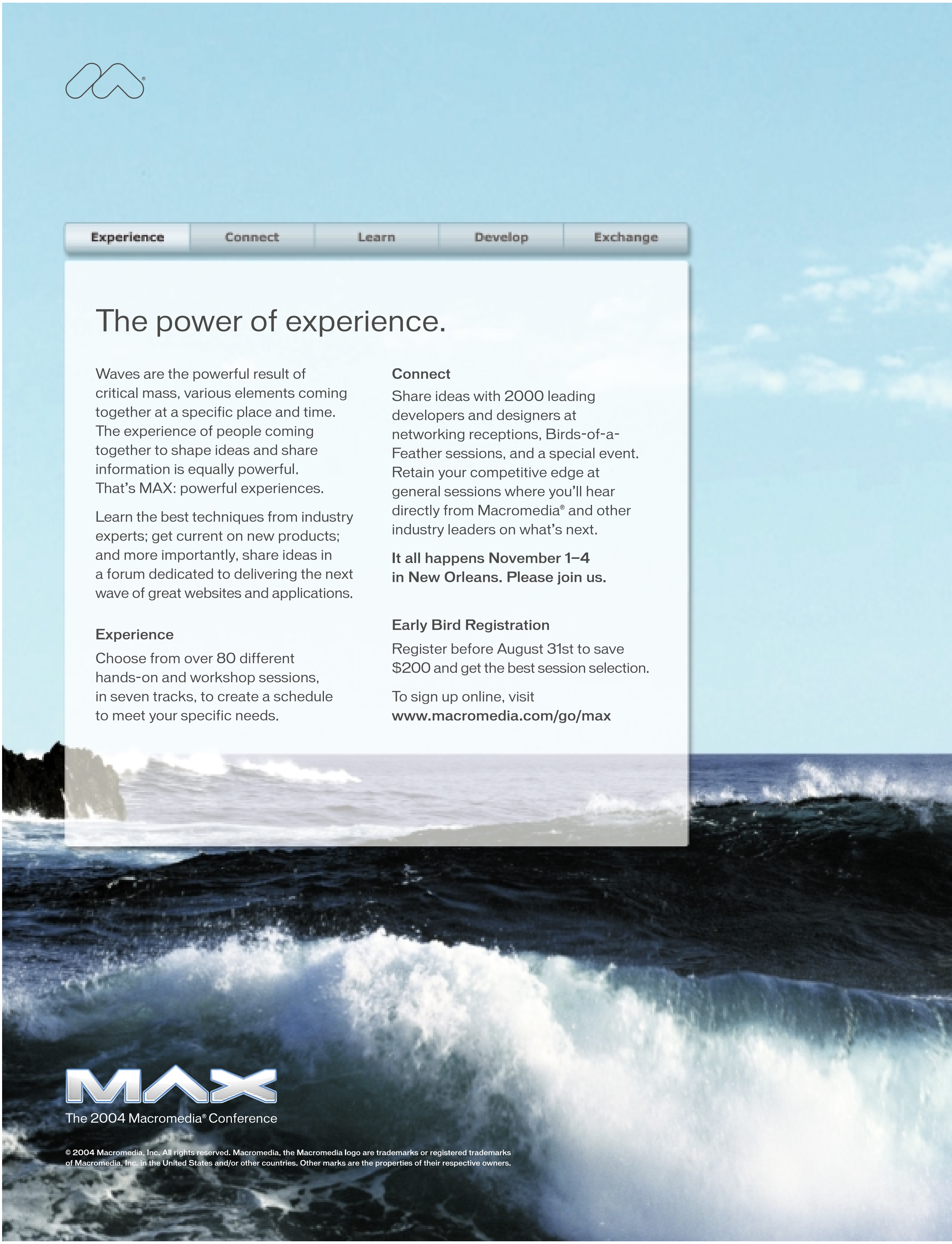
Listing 1

```
<cffunction name="rstoquery" output="false"
hint="returns a query from a Java ResultSet object">
  <cfargument name="resultset" type="any" required="true">

  <cfset var rs = resultset>
  <cfset var x = false>
  <cfset var col = false>
  <cfset var colnames = "">
  <cfset var tableData = false>

  <cfif isobject(resultset) and
findnocase("resultset",resultset.getClass().getName())>
    <cfset tabledata = resultset.getMetaData()>
    <cfloop index="x" from="1" to="#tableData.getColumnCount()#">
      <cfset colnames =
listappend(colnames,tableData.getColumnName(JavaCast("int",x)))>
    </cfloop>
    <cfset rs = querynew(colnames)>

    <cfloop condition="resultset.next()">
      <cfif resultset.getRow()>
        <cfset queryaddrow(rs)>
        <cfset x = rs.recordcount>
        <cfloop index="col" list="#colnames#">
          <cfset rs[col][x] =
resultset.getString(JavaCast("string",col))>
        </cfloop>
      </cfif>
    </cfloop>
  </cfif>

  <cfreturn rs>
</cffunction>
```

Listing 2

```
<cffunction name="jdbcType" output="false" returntype="string"
hint="returns the name or number for a given Java JDBC data type">
  <cfargument name="typeid" type="string" required="true">

  <cfset var sqltype = createobject("java","java.sql.Types")>
  <cfset var types = structnew()>

  <cfloop item="x" collection="#sqltype#">
    <cfset types[x] = sqltype[x]>
    <cfset types[sqltype[x]] = x>
  </cfloop>

  <cfreturn types[typeid]>
</cffunction>
```

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



Experience | Connect | Learn | Develop | Exchange

The power of experience.

Waves are the powerful result of critical mass, various elements coming together at a specific place and time. The experience of people coming together to shape ideas and share information is equally powerful. That's MAX: powerful experiences.

Learn the best techniques from industry experts; get current on new products; and more importantly, share ideas in a forum dedicated to delivering the next wave of great websites and applications.

Experience

Choose from over 80 different hands-on and workshop sessions, in seven tracks, to create a schedule to meet your specific needs.

Connect

Share ideas with 2000 leading developers and designers at networking receptions, Birds-of-a-Feather sessions, and a special event. Retain your competitive edge at general sessions where you'll hear directly from Macromedia® and other industry leaders on what's next.

It all happens November 1–4 in New Orleans. Please join us.

Early Bird Registration

Register before August 31st to save $200 and get the best session selection.

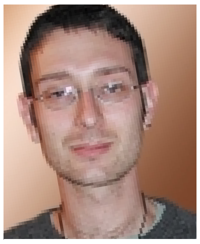
To sign up online, visit **www.macromedia.com/go/max**

MAX

The 2004 Macromedia® Conference

Fast Track to Flex

Macromedia Training 'Flexes' Its Muscles



By Simon Horwith

Hopefully by now you've heard all about Flex, the most recent addition to the Macromedia server product family. Hot on the tail of its release, Macromedia Training has released "Fast Track to Flex," a new course that introduces developers to developing rich Internet applications (RIAs) using the Macromedia Flex server.

If you don't already know it, Flex is a J2EE application that parses XML files (MXML files to be exact) and renders a Flash user interface based on the XML mark-up. Because the server renders Flash based on XML mark-up, no knowledge of Flash or the Flash authoring environment is required to use it. You don't have to have any knowledge of Flash or CF (or any other particular programming language) in order to take the class–the course prerequisite is that students be familiar with some basic object oriented programming ideas. Any CF developer used to using ColdFusion Components should do fine.

If you're familiar with ActionScript, JavaScript, Java, or any other programming language, so much the better. The course also assumes that students are comfortable with XML, but you don't need to be an XML expert – a simple understanding of tags and attributes and the rules of creating valid XML (every tag must have a closing tag, tags must be properly nested, and so on) is more than enough.

Like all other Macromedia Training offerings, the course units introduce students to new material via lecture, demonstrations, hands-on walkthroughs with the instructor, and independent labs. Fast Track to Flex is a two-day course broken-up into seven units:

- Introduction to the Course
- Introduction to Flex
- Interacting with Users
- Working with Flex Controls
- Working with External Data
- Managing Layouts
- Creating Navigation

Two days may not seem like much but, believe me, by the time you finish the second day you will not only feel comfortable creating Flex applications but will also have all of the tools you need to tackle and implement any Flex territory that's still unexplored. The remainder of this article is a summary of some but not all of the topics covered in the class.

What Does the Course Entail?

After being introduced to the class (format, objectives, topics, etc.), students are immediately introduced to Flex in Unit 2. In this unit they gain an understanding of the Flex Server Architecture, how Flash files are generated, how to configure the Flex Server, MXML, ActionScript, and they create their first simple Flex Application. The next unit introduces students to creating forms and form controls, how to bind form controls to each other, the user event model, using ActionScript to handle user events, and application architecture using Controller Classes.

Unit 4 introduces students to more of the form controls, embedding image objects in their Flex applications, creating custom form controls from Flash components, using formatting objects to format data input and output, supplying complex data to controls such as data grids, and to some common techniques for "skinning" controls with the Flex CSS API, ActionScript skins, and themes.

At this point, students know everything they need to know in order to build functional (and decent looking) RIA forms with Flex. The next unit teaches students the fundamentals of developing Flex applications that use the Model-View-Controller design pattern – a design pattern well-suited for Flex development. They also learn data validation and spend the majority of their time learning how to work with external data. Specifically, students learn how to work with XML data retrieved from a file on the server or from a URL, data retrieved from Web services, and data returned from Java Objects. They learn how to loop over and work with this complex data (XML, record sets, etc.), and how to configure the Flex Application Server to work with Java Objects and with Web services, and they spend a good deal of time learning how to work with data grid components (including the loading and use of complex data in data grids). The methods for working with external data sources (Web services, XML, and Java Objects) is done in the context of MVC best practices.

By this point, students will have a good understanding not only of how to build forms, but also how to integrate these forms with external data and how to do so with a "best practices" approach (using MVC to separate business logic from presentation layer and to reuse code).

The last two units of the Fast Track to Flex course deal primarily with the user interface again. Unit 6 introduces developers to the ins and outs of page layout containers. Students learn about panels and boxes (the two most simple and most common content layout containers) and the canvas and tile containers (more flexible and robust containers for content), and about positioning within containers. In addition, students also learn more about data validation and forms, and they explore and experiment with many of the visual effects (making things fade, zoom, move, and so on) that come with Flex.

Unit 7 deals with the user interface components that allow developers to layout the navigation for their applications. They learn how to do "forms based" navigation ("viewstack" development in Flex), and how to create and respond to interaction with link bars and tab bars and navigators (traditional menu-link and tab-based navigation). In addition, students learn how to create their own custom Flex components using MXML – very analogous to creating reusable UI/business logic objects in ColdFusion using Custom Tags.

Conclusion

Macromedia Training's newest course offering, Fast Track to Flex, is a very good place to start for any developer interested in building a solid foundation in Flex programming and in learning what Flex is really all about. This is one of those products that is definitely here to stay, and with the enhancements already promised in the next release as well as a little more time for companies to become better acquainted with its potential (not to mention the impending release of a Flex IDE), its presence in companies and the demand for Flex developers are sure to increase substantially.

If you or your employer are interested in evaluating Flex as a business solution; have already purchased Flex, and want to know how to begin developing RIAs using Flex; or just want to become more knowledgeable about what Flex really is and isn't as well as how to use it, then I highly recommend looking in to the Fast Track to Flex course.

Additional information about the class can be found at www.macromedia.com/support/training/instructor_led_curriculum/fast_track_flex.html, and you can find out which Macromedia Authorized Training Partner (MATP) facility offering the Fast Track to Flex class is closest to you at http://spectra15.macromedia.com/findaclass.cfm. Alternatively, you can contact your nearest MATP or e-mail training@macromedia.com for more information.

About the Author

Simon Horwith is co-technical editor of CFDJ, *and chief technology officer of eTRILOGY Ltd., a software development company based in London, England. Simon has been using ColdFusion since version 1.5 and is a member of Team Macromedia. He is a Macromedia Certified Advanced ColdFusion and Flash developer and is a Macromedia Certified Master Instructor. In addition to administering the* CFDJ *List mail list and presenting at CFUGs and conferences around the world, he has also been a contributing author of several books and technical papers.*

simon@horwith.com

Harnessing the Power of SQL Server Using Stored Procedures

Use of stored procedures with an n-tier architecture in your CFML code brings many benefits

By Grant Szabo

I've been developing ColdFusion applications since the early days of version 1.5. While a lot has changed for the better with the product since then, not a lot has changed – unfortunately – in terms of the lousy quality of the CFML code that I'm often asked to review or fix.

Most of the problems I see with CF apps have to do with poorly implemented Transact-SQL (T-SQL) and an overall failure to pull the T-SQL out of the middleware and push it down to the data tier where it really belongs. Whenever I have the opportunity to speak with developers who make this mistake, I often find out that they just plain don't know how to leverage the RDBMS effectively.

Grant: "So Joe, how come you aren't using stored procedures in this code?"

Joe Developer: "Well, I've never learned how to write stored procedures. I know it's the right way to do it but I'm just not sure how to get started."

This article will show you the fundamentals of writing stored procedures and tying them into your ColdFusion MX code. I'll focus on showing you how to do this using an n-tier architecture with ColdFusion MX.

Benefits of Using Stored Procedures

There are several reasons for using stored procedures instead of ad hoc queries. The two main ones relate to security and performance.

As far as security is concerned, using stored procedures instead of ad hoc queries in your CFML middleware ensures a higher level of security in your applications. Stored procedures allow for better data protection because they control how the data is accessed. Ad hoc queries are more susceptible to SQL Script injection attacks. When configuring the data source in the CF Administrator, only allow stored procedures, and make sure never to use the "sa" account to access SQL Server databases (see Figure 1).

With regard to performance, stored procedures generally perform a lot better than ad hoc queries. With CFMX 6.1 I've witnessed ad hoc queries perform roughly 50% better by doing nothing to the T-SQL other than cutting the ad hoc query out of the CFML middleware and pasting it into a new stored procedure. The final step involves replacing the T-SQL inside the <cfquery></cfquery> tag with "exec sp_the_new_proc". There are other ways to improve SQL Server performance beyond simply using stored procedures though. A great Web site ***CFDJ***'s own Simon Horwith made me aware of is SQL-Server-Performance.com (www.sql-server-performance.com), and a good book on the topic is *Microsoft SQL Server 2000 Performance Optimization Handbook* by Ken England.

Are there any downsides to using stored procedures? The most common argument is that by using stored procedures you're marrying your application to the RDBMS. Microsoft is doing some interesting stuff to debunk this using a Provider Model Pattern (see http://weblogs.asp.net/rhoward/archive/2003/11/07/36393.aspx), though my experience is that the argument is largely true. However, since 1995 I've never had to swap out an RDBMS for any of the hundreds of CF apps that I've written, other than upsizing MS Access to MS SQL Server.

Using n-tier architecture can minimize this blow if it ever happens to you. A lot of energy would be put into rewriting the T-SQL based stored procedures into PL/SQL (if switching from SQL Server to Oracle) but from a CF standpoint, you would just need to make minor adjustments to your business objects. By using a business object tier (read "CF Components"), all of your data access calls occur in your CFCs.

Smoke and Mirrors

For my first few years of developing CF applications I used Microsoft Access exclusively as my back-end database. I've found this to be pretty common among CF developers and indeed many CF developers have never moved beyond this familiar desktop database. The reality is that the majority of CF applications out there aren't of the scope or size that justify the cost of implementing an enterprise class RDBMS such as Microsoft SQL Server or Oracle. So, why tackle the tough stuff?

The reason for doing so is that Access doesn't support stored procedures and the CF/Access developer never learns how to write effective T-SQL. Most of the queries this type of developer writes are of the "SELECT * FROM TABLE WHERE ID=#ID#" variety. Even when such developers move to an enterprise-class database server, the queries continue to be done in-line (ad hoc), with no real improvement in the quality of the T-SQL. When I see this I just want to scream.

When I was learning how to write and call stored procedures from within ColdFusion I quickly learned that it was a lot of smoke and mirrors – it's just not that hard to do once you understand the basics. Perpetuating the smoke and mirrors is the fact that there are many complicated-sounding terms to be concerned about, like Attributes, Parameters, Cursors, @@IDENTITY, and so on. (I'll try to intentionally pepper this article with some of these big college-sounding words in order to demystify them.) Another drawback is that there aren't any CF books that I'm aware of that are dedicated to this topic. So you are obliged instead to glean what information you can from existing titles: Microsoft's decidedly ASP-, ASP.NET-, and ADO.NET-focused resources, and the SQL Server Books Online. It helps to have a friend who can help you out, but if you don't have a special friend, take heart, you can do it…and I'm about to show you how.

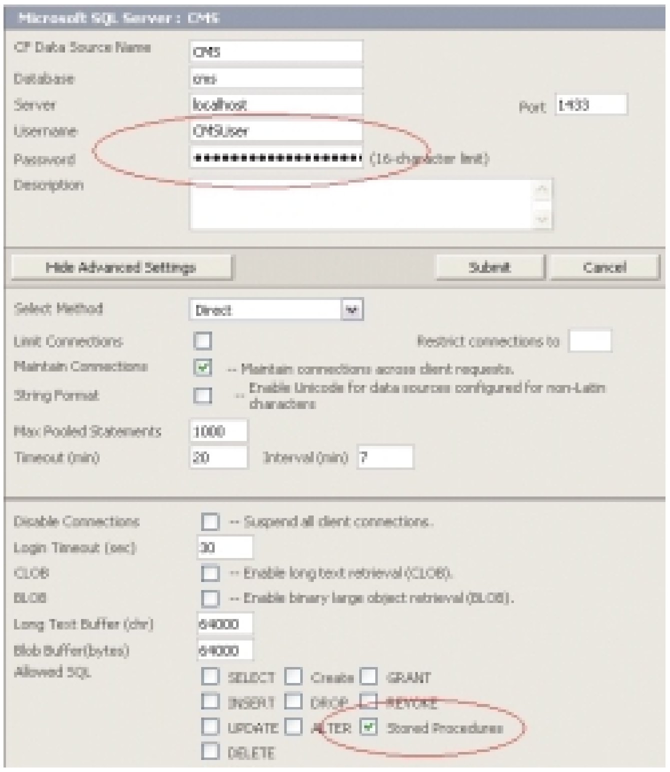
First, Get the Tools

If you don't have a copy of Microsoft SQL Server, you should immediately buy the $50 developer edition. Microsoft lowered the price from several hundred dollars to just $50 within the last year; if you didn't know this already, now you have no excuse. Buy it and install it on the workstation or laptop where you do your programming work.

A Microsoft SQL Server purchase (any edition) includes SQL Enterprise Manager and SQL Query Analyzer, two utilities that can be used to write stored procedures. There are other third party products available as well. One free product that I've looked at is Toad for SQL Server (available for free download at www.toadsoft.com/toadsqlserver/toad_sqlserver.htm). My personal preference is SQL Query Analyzer for two main reasons:

1. It's easy to test stored procedures from within SQL Query Analyzer. You right-click on a stored procedure and choose OPEN. A property dialog box opens prompting you to enter all the values for your parameters.
2. In addition to seeing all of your tables listed out alphabetically, it is possible to expand each one (by clicking the "+" symbol next to the table name) so that the column name, datatype, and size can be seen for each attribute. This is very helpful as you will need to know this information when you are writing a stored procedure. See Figure 2.

There are a few other reasons why I like to use SQL Query Analyzer. What's important isn't the tool you ultimately use, but that you are comfortable with the tool you choose, and in my experience SQL Query Analyzer is pretty easy to use. In this article I will use SQL Query Analyzer, though you can use whatever tool you're most comfortable with to accomplish the same tasks.


Microsoft SQL Server : CMS

CF Data Source Name	CMS
Database	cms
Server	localhost — Port 1433
Username	CMSUser
Password	******************** (16-character limit)
Description	
Select Method	Direct
Limit Connections	☐ — Restrict connections to
Maintain Connections	✓ -- Maintain connections across client requests.
String Format	☐ -- Enable Unicode for data sources configured for non-Latin characters
Max Pooled Statements	1000
Timeout (min)	20 — Interval (min) 7
Disable Connections	☐ -- Suspend all client connections.
Login Timeout (sec)	30
CLOB	☐ -- Enable long text retrieval (CLOB).
BLOB	☐ -- Enable binary large object retrieval (BLOB).
Long Text Buffer (chr)	64000
Blob Buffer(bytes)	64000
Allowed SQL	☐ SELECT ☐ Create ☐ GRANT ☐ INSERT ☐ DROP ☐ REVOKE ☐ UPDATE ☐ ALTER ✓ Stored Procedures ☐ DELETE

Figure 1: SQL Server datasource configuration

Clearing the Smoke and Shattering the Mirrors

OK, so now you have the tools you need to begin writing stored procedures. What now? Let's do this using n-tier architecture. This means we do our best to divvy up our code as follows:

1. ***Data Tier:*** This is the lowest level of our application. It's where our data lives as well as our API for interacting with our data. Business objects in the Business Objects tier will interact with this API exclusively.
2. ***Business Object Tier:*** This is the middle layer of our application. It's where we encapsulate business rules, make calls to our data tier, and create an API for our presentation tier to interact with.
3. ***Presentation Tier:*** Otherwise known as the User Interface (UI), this is where we'll write our typical HTML and CFML code.

Step One – Create your Entities (Also Known As Tables)

After you understand your project requirements, begin by creating your tables in SQL Enterprise Manager. All of my tables start with the acronym "tbl" preceding the table name. Read up on SQL Server datatypes using SQL Server Books Online. Books Online is Microsoft's help file system for SQL Server and SQL – an invaluable tool for anyone using the database. You can press the F1 key from within SQL Enterprise Manager or SQL Query Analyzer to easily start Books Online. The most common datatypes that you will likely use are int, char, varchar, and datetime. Use Hungarian notation for your table attributes. Table 1 shows some examples using a table named tblOrder so you can see what Hungarian notation looks like.

Hungarian notation allows you to see the datatype your table attributes are expecting without having to constantly reference the database server. When using CFSTOREDPROC, you'll need to specify the datatype of your stored procedure

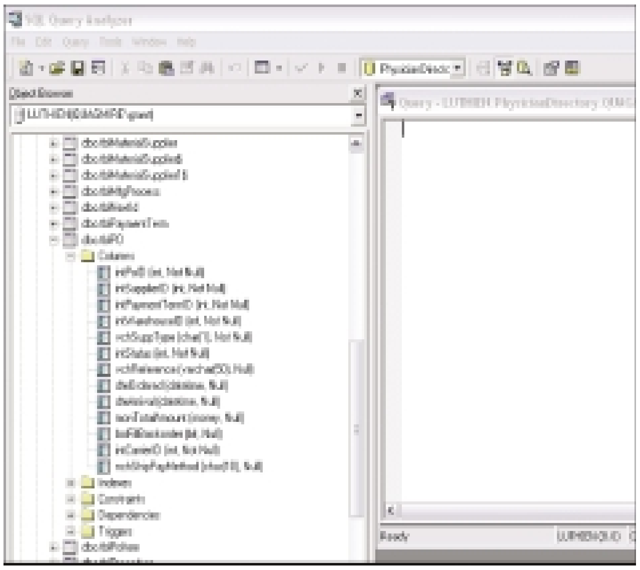


Figure 2: SQL query analyzer

DATATYPE	BAD COLUMN NAME	GOOD COLUMN NAME
Integer	ID	intOrderId
Varchar	orderNote	vchOrderNote
DateTime	orderDate	dteOrderDate
Money	grandTotal	monGrandTotal

Table 1: Using Hungarian notation

parameters. If you use Hungarian notation, this job becomes a lot easier.

Please note: if you create this table in MS SQL Server, the intOrderId attribute should be made the primary key and the Identity property should be set to YES. The Identity property is the MS Access equivalent of Autonumber in case you are wondering.

Step Two – Write your Append, Update, and Delete Stored Procedures

Most, if not all, tables in your database will require these three stored procedures at a base minimum. There are code generators available to automate this task, but while you are learning, you should hand-code them. Do try to comment your stored procedures when possible. A single line comment uses two hyphens preceding the comment or, for larger comment blocks, the familiar "/* comment here */" notation is used.

Here is the syntax you will follow when writing a stored procedure with Microsoft SQL Server:

```
CREATE|ALTER PROCEDURE [OWNER].[PROCEDURE NAME]

[PARAMETER LIST]

AS

[STOREDPROC BODY]
```

I recommend you not specify the [OWNER] in your stored procedures, but have included it here for completeness.

For all of my stored procedures, I use the following naming convention. You should use something similar so that the functionality of any given stored procedure is readily identifiable just by reading its name:

1. Begin all procs with "sp" for stored procedure.
2. Immediately after the sp, in uppercase, I place the acronym for my application.
3. Underscore, followed by an App for Append, Upd for Update, Del for Delete, Sel for Select, or Utl for Utility, followed by another underscore.
4. The name of the table I am working with, without the "tbl".
5. For Select procs, I generally follow this by another underscore and some form of short description, such as "spGS_Sel_Order_getByOrderId". For Utility procs where more than one table is being accessed, or if I am using a cursor to do some kind of iterative operation, I name the proc as descriptively as possible, such as "spGS_Utl_ProcessOrderByUUID".

Seapine Software™
changing the world of software development

Complete source code and asset management in Dreamweaver MX—now possible with Surround SCM.

Dreamweaver users know a beautiful Web-based product is only skin deep. Underneath, it's a tangle of hundreds or thousands of ever changing source files. Without a good development process and strong tools, bad things happen. Surround SCM can help.

Surround SCM lets you...

Track multiple versions of your source files and easily compare and merge source code changes.

Check out files for exclusive use or work in private workspaces when collaborating on a team.

Automatically notify team members of changes to source files—push changes through your organization.

View complete audit trails of which files changed, why, and by whom.

Associate source code changes with feature requests, defects or change requests (requires additional purchase of TestTrack Pro).

Remotely access your source code repository from Dreamweaver MX.

Surround SCM adds flexible source code and digital asset control, powerful version control, and secure remote file access to Dreamweaver MX. Whether you are a team of one or one hundred, Surround SCM makes it easier to manage your source files, letting you focus on creating beautiful Web-based products.

Features:

Complete source code and digital asset control with private workspaces, automatic merging, role-based security and more.

IDE integration with Dreamweaver MX, JBuilder, Visual Studio, and other leading Web development tools.

Fast and secure remote access to your source files—work from anywhere.

Advanced branching and email notifications put you in complete control of your process.

External application triggers let you integrate Surround SCM into your Web site and product development processes.

Support for comprehensive issue management with TestTrack Pro—link changes to change requests, bug reports, feature requests and more.

Scalable and reliable cross-platform, client/server solution supports Windows, Linux, Solaris, and Mac OS X.

Want to learn more?

Download Seapine's just-released white paper, **Change Management and Dreamweaver**, to discover tips, tricks and best practices for achieving full software configuration functionality. Visit www.seapine.com/whitepaper.php and enter code WM0604 to receive your copy today.

www.seapine.com
1-888-683-6456

Surround SCM Source Control

TestTrack Pro Defect Management

QA Wizard Automated Testing

macromedia ALLIANCE PARTNER

Here are simple Append, Update, and Delete procs. To create these procedures, I would type them in just as you see them here, one at a time, and press CTL-E when complete to execute (compile) the stored procedure. Executing compiles the proc and makes it available for future use. If any errors occur during compiling, you'll be alerted and given some clues as to what you did wrong so you can fix the problem. Think about this for a second. You're writing and compiling your database objects in advance. SQL Server won't have to do this at runtime (for example, if you had been using ad hoc queries). This is a good thing. Moreover, you can think of your stored procedures as levers or buttons that you press from ColdFusion when you want some action to occur with your data. Think of your stored procedures as your API (application programming interface) to your data tier – because that's exactly what they are (see Listing 1).

Now we have written the mandatory Append, Update, and Delete procs. How about a SELECT proc to return the list of orders descending on order date:

```
CREATE PROCEDURE spGS_Sel_Order_getAll

AS

SELECT intOrderId,
     dteOrderDate,
  vchOrderNote,
  monGrandTotal
FROM        tblOrder
ORDER BY dteOrderDate DESC
```

CFSQLTYPE	MS Access	MS SQL
CF_SQL_BIGINT		bigint
CF_SQL_BIT	Yes/No	bit
CF_SQL_CHAR		char, nchar
CF_SQL_DATE		
CF_SQL_DECIMAL		numeric, decimal
CF_SQL_DOUBLE		double, float
CF_SQL_FLOAT		double, float
CF_SQL_IDSTAMP		timestamp
CF_SQL_INTEGER	AutoNumber	int
CF_SQL_LONGVARCHAR	Memo	text
CF_SQL_MONEY	Currency	money
CF_SQL_MONEY4		smallmoney
CF_SQL_NUMERIC	Number	numeric, decimal
CF_SQL_REAL		real
CF_SQL_REFCURSOR		cursor
CF_SQL_SMALLINT		smallint
CF_SQL_TIME		
CF_SQL_TIMESTAMP	Date/Time	datetime, smalldatetime
CF_SQL_TINYINT		tinyint
CF_SQL_VARCHAR	Text	varchar, nvarchar, uniqueidentifier

Table 2: CFSTOREDPROC – MS SQL mappings

And, one select proc to get a specific order by ID if the user clicks on a specific order in the list:

```
CREATE PROCEDURE spGS_Sel_Order_getByOrderId

@intOrderId         int

AS

SELECT intOrderId,
     dteOrderDate,
  vchOrderNote,
  monGrandTotal
FROM        tblOrder
WHERE intOrderId = @intOrderId
```

Step Three – Write your CFC

You've now completed writing your data-tier interface for tblOrder. It's time to write the business object for this API.

We do this in ColdFusion using CFCs. Start your development IDE of choice (I like CFStudio 5 or Homesite+) and create your CFC. Remember, your business object is the bridge between the presentation layer (the User Interface) and the data tier. Using Web services jargon, your business object will create an API for your UI to consume while at the same time consuming the interface you just created in your stored procedures. I'm not going to show you how to properly architect your CFC here as that is outside the scope of this article. Rather, I'll show you how to call upon the stored procedures we created in Step Two (see Listing 2). Please note that this CFC doesn't use any error handling, which is bad, but – again – that's outside the scope of this article.

There are two important points to remember when using CFSTOREDPROC. First, your CFPROCPARAM tags absolutely must be in the exact same order as your parameters are specified in your stored procedure. Not doing this will create problems and you will often get database exceptions because data types aren't lining up properly. If you get these kinds of exceptions, carefully compare the ordering of your CFPROCPARAM tags and your [PARAMETER LIST] in your stored procedure. Often you will find you missed an attribute completely or just put something in the wrong order.

Second you must select the proper CFSQLTYPE property in your CFPROCPARAM tags. Please consult Table 2 for the CFSQLTYPE to SQL 2000 equivalents. I've included an MS Access column so that you can see the MS SQL Server equivalents for MS Access as well.

It's important to mention that you can execute stored procedures using CFQUERY instead of CFSTOREDPROC. Here's the CFQUERY equivalent to the CFSTOREDPROC code above used in the fetchOrderByOrderId CFFUNCTION:

```
<CFQUERY name="RS1" datasource="#this.DSN#">
   EXEC spGS_Sel_Order_getByOrderId #arguments.orderId#
</CFQUERY>
```

Which to use? Unfortunately there's no good rule of thumb and there's very little written on the topic. I've found in some

LinuxWorld® Conference & Expo

CONFERENCE: August 2 – 5, 2004 EXPO: August 3 – 5, 2004

MOSCONE CENTER • SAN FRANCISCO, CA

Where OPEN MINDS Meet

LinuxWorld Conference & Expo is the #1 venue for decision-makers and influencers to discover real business solutions for real business problems, learn how Linux is accelerating as a total enterprise solution, and understand how it can be applied effectively to save their companies time and money.

Attend LinuxWorld and...

- Realize Linux as more than an operating system, but as **a world of applications**
- Explore interoperability issues and opportunities in **open source and proprietary environments**
- Stay on the cusp of **emerging technologies** and the acceleration of open source adoption in enterprise computing
- Review the latest open source initiatives, their deployment and successes to help **make informed decisions** for your company
- Hear compelling keynotes given by top executives from BEA, HP, IBM, Oracle, and Red Hat

WWW.LINUXWORLDEXPO.COM

Register Online With Priority Code: D2601

CORNERSTONE SPONSOR

ORACLE

PLATINUM SPONSORS

Computer Associates DELL hp invent IBM intel Novell Sun microsystems

IDG WORLD EXPO

cases that CFQUERY performs better than CFSTOREDPROC. We're talking milliseconds here, folks, not noticeable differences generally speaking. I've also found that CFSTOREDPROC doesn't always work reliably in situations where I have 40 or more input parameters. But, if your stored procedure returns more than one record set, CFSTOREDPROC is your only option. I tend to use CFSTOREDPROC almost exclusively in my code because I find it to be more readable as a developer. CFPROCPARAM specifies my datatypes and whether the parameter direction is IN, OUT, or both. I find it easier to debug CFSTOREDPROC for this reason.

Step Four - Write Your Presentation Layer Code (UI)

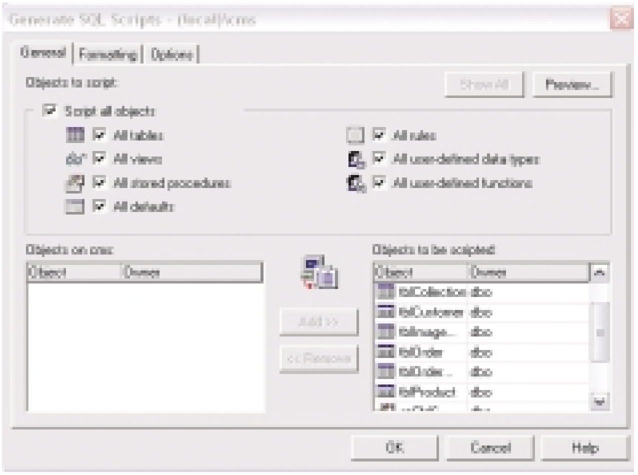
Now that you have all of the pieces complete, you can create your presentation layer. Invoke your business object using <CFINVOKE> or using CreateObject() inside a CFSCRIPT block such as you see at Listing 3.

Step Five - Script Your Database

This step isn't essential to getting your stored procedures to work, but it is essential for protecting your time investment in your T-SQL code. Scripting the database is done with SQL Enterprise Manager. This operation will script all of the database objects that you specify into a big T-SQL script that you can then save off in your source code repository. This is really important if you ever need to recreate your database on a new database server. Note that scripting your database doesn't preserve the actual data in the tables. That's what backups are for. To access the Generate SQL Script feature of SQL Enterprise Manager, right-click on a database in the tree view, choose ALL TASKS, and then choose Generate SQL Script. Figure 3 depicts this dialog box.

Error Handling in Your Stored Procedures

With a Delete operation you often need to check to make sure that there will be no orphaned child records after the delete. If there are, generally a warning is displayed to the user and the operation is halted until all of the child records are removed, at which point the user can try deleting again. Without using RAISERROR, this takes two queries, one to check for child records, and another to do the deletion if there aren't any. Using RAISERROR, you can eliminate the extra query and handle it all in a single stored procedure (see Listing 4).


Generate SQL Scripts - (local)\cms
General | Formatting | Options
Show All
Preview...
Add >>
<< Remove
OK
Cancel
Help

Objects to script:

- [✓] Script all objects
  - [✓] All tables
  - [✓] All views
  - [✓] All stored procedures
  - [✓] All defaults
  - [✓] All rules
  - [✓] All user-defined data types
  - [✓] All user-defined functions

Objects on cms:

Object	Owner

Objects to be scripted:

Object	Owner
tblCollection	dbo
tblCustomer	dbo
tblImage...	dbo
tblOrder	dbo
tblOrder...	dbo
tblProduct	dbo

Figure 3: SQL Enterprise Manager generated script

Here's a cffunction I wrote for cleaning up the raiserror string so that it can be displayed nicely to the end user:

```
<cffunction name="raiseErrorHandler" access="public"
returntype="string">
<cfargument name="errorMessage" type="string" required="yes">
<cfscript>
var rVal = "";
if(REFind("\[DBERROR]", arguments.errorMessage, 1)) {
rVal = REReplace(arguments.errorMessage, "\[(Macromedia|SQLServer JDBC
Driver|SQLServer)\]", "" , "ALL");
rVal = REReplace(rVal, "\[DBERROR]", "", "ALL");
}
return rVal;
</cfscript>
</cffunction>
```

You'll need to wrap a try/catch around your database call (around the cfstoredproc statement) and in the event of a database catch, feed the cfcatch.detail to this function.

Using Transactions

SQL Server 2000 also supports the use of Transactions. A transaction is a single unit of work. Situations that require transactions typically include scenarios where you are moving data from one table to another (and if anything goes wrong the whole job should be voided). If you're using CFTRANSACTION, you don't need to use transactional processing in your stored procedures. When you need to implement a transaction, give some thought as to how you want to do it and how you think it will best perform – in the middleware (e.g., in your CFML) or in the data tier. SQL Server supports several isolation levels and you should check SQL Server Books Online to ensure you are using the correct isolation level. READ COMMITTED is the SQL Server default if you don't specify the transaction isolation level. Here's how you implement a transaction in a stored procedure (this T-SQL code comes after the AS keyword in the stored procedure):

```
-- set the isolation level if there are any SELECT statements used
below, otherwise this line isn't needed
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

BEGIN TRANSACTION

/* perform data operations HERE that must all be successfully complet-
ed as a single unit of work */

-- check to see if an error condition was encountered and if so, roll-
back
IF @@ERROR != 0
    BEGIN
```

```
/*ROLLBACK TRANSACTION erases all data modifications made since the
start of the transaction and frees resources held by the transaction.
*/
    ROLLBACK TRANSACTION
    END
ELSE
    BEGIN
-- No errors, commit the transaction
COMMIT TRANSACTION
    END
```

Developer, Know Thy Database

You should now have an understanding of the basics when it comes to implementing stored procedures in your CFMX application. If you have a chance to work with an enterprise-class CF application, spend some time reading the stored procedures and enhancing your knowledge. As your knowledge increases, you'll probably want to spend some time reading about cursors, functions, temporary tables, table variables, and transactions. A SQL Server DBA I once spoke with told me he reads one stored procedure per day from the MS SQL Master database. This is a database that's installed by SQL Server and drives much of the functionality within it. That's a great tip to follow.

Summary

In closing, I hope this article has at least piqued your interest in learning more about stored procedures. The benefits of using stored procedures with an n-tier architecture in your code include:

1. ***Better performance:*** Procs are precompiled instead of needing to be compiled at runtime by the SQL Server.
2. ***With n-tier architecture your code is easier to read and to maintain:*** No more digging through spaghetti code trying to find a buggy query.
3. ***Encapsulation of complex data tasks:*** Stored procedures can perform multiple data operations, one after the other within the same proc. Retrieving the @@IDENTITY after, doing an append operation, as shown in the Append procedure above, is an example of this. Instead of having two <cfquery> operations in your middleware code, one to insert and the next to get the latest ID, you have one stored procedure to handle it all. By using Cursors, it is possible to perform iterations within your stored procedure, and SQL Server 2000 supports Functions as well. These are areas that are worthwhile to read up on to fully exploit the power of stored procedures. (See spGS_Utl_ProcessOrder for an example of using a single stored procedure to perform multiple data operations; see Listing 5). A forthcoming advanced topic article is planned that will discuss spGS_Utl_ProcessOrder in greater detail, as well as get into the finer points of database transactions.
4. ***Push button capability:*** Call the proc using the cfstoredproc tag (or cfquery) and it does what it is programmed to

HostMySite.com™

Built for ColdFusion Pros by ColdFusion Pros

- 24 / 7 / 365 Phone Support
- 99.9+% Uptime
- Macromedia Alliance Partner
- "Full Control" Panel
- CFMX 6.1 or CF 5.0
- SQL Server 2000 or 7.0
- Custom Tags Welcome

plans from $8.95 / mo.

FREE Domain Name*
FREE Setup
FREE 2 Months

Visit www.HostMySite.com/cfdj for:
2 Months Free
FREE Setup and FREE Domain Name
*on any annual shared hosting plan

call today 877•248•HOST (4678)

do; just like pulling a lever and having your butler show up with your dinner. Well, not exactly, but you get the idea.

Stored procedures are very powerful ways to quickly and efficiently handle data operations. Time spent exploring stored procedures in depth while increasing your knowledge of Transact-SQL is well worth it.

About the Author

Grant Szabo is a senior application developer for Global Cloud, Ltd. (www.globalcloud.net), where he delivers Microsoft .NET and ColdFusion solutions for Global Cloud's clients. Grant worked for Allaire (later Macromedia), as director, worldwide professional services for nearly two years in 2000-2001. He is certified in CF5 and CFMX and holds numerous Microsoft certifications including MCSD, MCDBA, MCSE, and MCSA.

grant@quagmire.com.

Listing 1:

```
CREATE PROCEDURE spGS_App_Order

@vchOrderNote varchar(20),
@monGrandTotal money
@intOrderId int OUTPUT /* lets have this proc return the new intOrderId
to us */

AS

/* get the system date automatically, forget about having to use
CreateODBCDate() forever! */

DECLARE @dteOrderDate datetime
SET @dteOrderDate = GETDATE()

INSERT INTO tblOrder (vchOrderNote,
                      dteOrderDate,
                       monGrandTotal)
VALUES (@vchOrderNote,
       @dteOrderDate,
           @monGrandTotal)

/* Use @@IDENITY to return the new primary key value of the order that
was just inserted */
SELECT @intOrderId = @@IDENTITY
```

```
CREATE PROCEDURE spGS_Upd_Order

@intOrderId int,
@vchOrderNote varchar(20),
@monGrandTotal money

AS

DECLARE @dteOrderDate datetime
SET @dteOrderDate = GETDATE()


UPDATE tblOrder
SET       vchOrderNote = @vchOrderNote,
    dteOrderDate = @dteOrderDate,
   monGrandTotal = @monGrandTotal
WHERE intOrderId = @intOrderId
```

```
CREATE PROCEDURE spGS_Del_Order

@intOrderId int

AS

/* Alternatively, you may wish to modify tblOrder to include a bit field
named 'bolActive' and have spGS_Del_Order update bolActive, setting it
to 'false' in the event of a deletion. This creates a deleted orders
archive, allowing an accidentally deleted order to be recovered.
Alternatively, a deleted order could be inserted into another table and
then deleted from tblOrders. These actions are beyond the scope of this
article but are mentioned here so you are aware of alternate possibili-
ties. */

DELETE FROM tblOrder
WHERE intOrderId = @intOrderId
```

Listing 2:

```
<cfcomponent displayname="order" hint="order business object">

<cfscript>
//pseudo-constructor
this.DSN = request.dsn;
</cfscript>

<!— note how the output parameter is handled and returned by this method
‡
<cffunction name="appendOrder" access="public" returntype="numeric">
<cfargument name="vchOrderNote" type="string" required="Yes">
<cfargument name="monGrandTotal" type="numeric" required="yes">
<cfstoredproc procedure="spGS_App_Order" datasource="#this.DSN#">
<cfprocparam type="In" cfsqltype="CF_SQL_VARCHAR"
dbvarname="@vchOrderNote" value="#arguments.vchOrderNote#" null="No">
<cfprocparam type="In" cfsqltype="CF_SQL_MONEY"
dbvarname="@monGrandTotal" value="#arguments.monGrandTotal#" null="No">

<cfprocparam type="Out" cfsqltype="CF_SQL_INTEGER" variable="intOrderid"
dbvarname="@intOrderId" null="No">
</cfstoredproc>
<cfreturn intOrderId>
</cffunction>

<cffunction name="updateOrder" access="public" returntype="void">
<cfargument name="intOrderId" type="numeric" required="Yes">
<cfargument name="vchOrderNote" type="string" required="Yes">
<cfargument name="monGrandTotal" type="numeric" required="yes">
<cfstoredproc procedure="spGS_Upd_Order" datasource="#this.DSN#">
<cfprocparam type="In" cfsqltype="CF_SQL_INTEGER" dbvarname="@intOrderId"
value="#arguments.intOrderId#" null="No">
<cfprocparam type="In" cfsqltype="CF_SQL_VARCHAR"
dbvarname="@vchOrderNote" value="#arguments.vchOrderNote#" null="No">
<cfprocparam type="In" cfsqltype="CF_SQL_MONEY"
dbvarname="@monGrandTotal" value="#arguments.monGrandTotal#" null="No">

</cfstoredproc>
</cffunction>

<cffunction name="deleteOrder" access="public" returntype="void">
<cfargument name="intOrderId" type="numeric" required="Yes">
<cfstoredproc procedure="spGS_Del_Order" datasource="#this.DSN#">
<cfprocparam type="In" cfsqltype="CF_SQL_INTEGER" dbvarname="@intOrderId"
value="#arguments.intOrderId#" null="No">
</cfstoredproc>
</cffunction>

<cffunction name="fetchOrderList" access="public" returntype="query">
<cfset var RS1 = "">
<cfstoredproc procedure="spGS_Sel_Order_getAll" datasource="#this.DSN#">
<cfprocresult name="RS1">
</cfstoredproc>
<cfreturn RS1>
</cffunction>

<cffunction name="fetchOrderByOrderId" access="public"
returntype="query">
<cfargument name="intOrderId" type="numeric" required="yes">
<cfset var RS1 = "">
<cfstoredproc procedure="spGS_Sel_Order_getByOrderId"
datasource="#this.DSN#">
<cfprocparam type="In" cfsqltype="CF_SQL_INTEGER" dbvarname="@intOrderId"
value="#arguments.intOrderId#" null="No">
<cfprocresult name="RS1">
</cfstoredproc>
<cfreturn RS1>
</cffunction>

</cfcomponent>
```

Listing 3:

```
<cfscript>
//Retrieve a record set of orders, descending by date

//instantiate the order object
objOrder = CreateObject('component', 'GS.components.order');

//invoke the fetchOrderList() method
RS1 = objOrder.fetchOrderList();
</cfscript>
```

```
<cfdump var="#RS1#">


<cfscript>
//Append an Order and retrieve the new order id

//instantiate the order object
objOrder = CreateObject('component', 'GS.components.order');

//specify variable name to hold output parameter, invoke appendOrder()
method
intOrderId = objOrder.appendOrder(form.vchOrderNote, form.monGrandTotal);
writeOutput("New Order Id = " & intOrderId);
</cfscript>
```

Listing 4:

```
CREATE  PROCEDURE spGS_Del_Category

@intCategoryId      int

AS

IF EXISTS (SELECT intCategoryId
    FROM tblProduct
    WHERE intCategoryId = @intCategoryId)

  BEGIN

/* the string [DBERROR] will be checked for in a CF Function and if it
is found, we know that the error was raised by a stored procedure */

/* the integers at the end of the RAISERROR statement are SEVERITY and
STATE respectively. See SQL Server Books Online for more information on
setting these values. STATE is an arbitrary integer between 1 and 127
that represents information about the invocation state of the error. The
values depicted below are generally acceptable. The severity of 11 indi-
cates that this is an error that the user can correct. */

RAISERROR('[DBERROR]This category has products assigned. Remove or re-
assign the products before trying to remove this category. Deletion
fails.',11,1)

  END

ELSE
  BEGIN

  DELETE FROM tblCategory
  WHERE intCategoryId = @intCategoryId

  END
```

Listing 5:

```
ALTER  PROCEDURE spGS_Utl_ProcessOrder

@nchUUID            char(35),
@intCustomerId               int,
@intResult                   int,
@nchAuthCode                 char(6),
@monAmtCharged      money,
@monSubTotal                 money,
@monTax             money,
@monShipping                 money,
@nchCvv2Match                char(1),
@bolMember                   bit

AS

/* Purpose: rolls an authenticated order over from staging into non stag-
ing tables. This proc does not return
a record set. It just takes input parameters and based on these values,
attempts to roll the order over the shopping cart
(tblStagingOrderProduct) to the order tables (tblOrder and
tblOrderProduct)
  Developer: G Szabo
  Created: 6/1/2004
*/

-- make sure product pricing information isn't changed in the middle of
this rollover
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRANSACTION

DECLARE  @intOrderId         int,
         @dteOrderDate       datetime,
         @intProductId       int,
         @intQuantity        int,
         @monPricePerUnit money

-- set the order date
SET @dteOrderDate = GETDATE()

-- get the next order ID from tblNextNum (you can use identity columns
instead of this method if you wish)
EXEC spCMS_Utl_NextNum @intOrderId OUTPUT

-- insert the order header
INSERT INTO tblOrder (        intOrderId ,
                     intCustomerId          ,
                     dteOrderDate           ,
                     intResult  ,
                     nchAuthCode            ,
                     monAmtCharged          ,
                     monSubTotal            ,
                     monTax                 ,
                     monShipping            ,
                     bolShipped ,
                     nchCvv2Match           )
VALUES    (          @intOrderId                      ,
                     @intCustomerId                   ,
                     @dteOrderDate                    ,
                     @intResult             ,
                     @nchAuthCode                     ,
                     @monAmtCharged         ,
                     @monSubTotal                     ,
                     @monTax                ,
                     @monShipping                     ,
                     0                                ,
                     @nchCvv2Match                    )

-- insert the order items (children)
DECLARE csrRecord CURSOR LOCAL SCROLL STATIC FOR
SELECT intProductId,
   intQuantity,
FROM      tblStagingOrderProduct
WHERE nchUUID = @nchUUID

OPEN csrRecord
FETCH NEXT FROM csrRecord INTO @intProductId, @intQuantity, @intMKEY

/* this operation will iterate over each item the customer placed in the
staging table (the shopping cart) and insert each row into
tblOrderProduct (live order items table). Alternatively a temp table
could be used instead of a cursor. Since there generally aren't that many
items in a cart, a cursor is being used. */
WHILE @@FETCH_STATUS = 0

  BEGIN

  -- if the user is a member, they get a discount, check membership sta-
tus and apply correct price
  IF(@bolMember = 1)
        BEGIN
        SELECT @monPricePerUnit = monMemberPrice
        FROM       tblProduct
        WHERE intProductId = @intProductId
        END
  ELSE
        BEGIN
        SELECT @monPricePerUnit = monNonMemberPrice
        FROM       tblProduct
        WHERE intProductId = @intProductId
        END

  -- perform the insert operation
  INSERT INTO tblOrderProduct ( intOrderId         ,
  intProductId      ,
  intQuantity       ,
                                          bolShipped ,
                                          monPricePerUnit )
  VALUES (                                @intOrderId,
                                          @intProductId,
                                          @intQuantity,
                                          0,
                                          @monPricePerUnit)

  FETCH NEXT FROM csrRecord INTO @intProductId, @intQuantity, @intMKEY

  END

  -- clean up
  CLOSE csrRecord
  DEALLOCATE csrRecord

IF @@ERROR !=0
  BEGIN
  ROLLBACK TRANSACTION
  RAISERROR('[DBERROR] An error occurred in the order rollover transac-
tion.',11,1);
  END
ELSE
  BEGIN
  COMMIT TRANSACTION
  END
```

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

A Wedding Invitation

A perfect marriage between ColdFusion and Java

By Hal Helms

If you missed this year's CFUN conference (June 26–27), you missed a lot. In addition to the great time spent meeting and talking with other ColdFusion programmers, Ben Forta gave a keynote demo of the next version of ColdFusion, code-named "Blackstone".

I haven't been this excited about the release of a version of ColdFusion in quite some time. Blackstone has new features for using Flash to produce very sophisticated, very user-friendly forms with advanced features such as tabs and accordions. It adds excellent support for producing PDF files from native ColdFusion code, and it introduces a new, very powerful report writer. And of course, it does all this with the trademark ease of programming for which we've come to depend on ColdFusion. It will, in short, make you a coding hero.

This makes ColdFusion the best presentation language available and this is important – very important – because to the users, the user interface *is* the application. I have been puzzling for some time over the question of where ColdFusion fits in the enterprise space increasingly dominated by the J2EE and .NET platforms, and with Ben's presentation I think I see how perfect a marriage ColdFusion is with Java.

Java is the proverbial 800-pound gorilla in the enterprise computing space. It runs on virtually every processor (given its "write once; run anywhere" ability) and is used for everything from running Mars rovers to cellphones to gas station pumps. But Java is primarily a server-side language, which is where it excels. Adoption of Java applets on the other hand, has slowed to a crawl.

ColdFusion – particularly the Blackstone version – excels at providing the presentation layer, but is much weaker than Java on the server side, where it lacks such Java features as constructors, interfaces, abstract classes, overloading, and a null object – all undergirded by Java's strong data typing that virtually eliminates runtime exceptions.

In this article, I want to demonstrate how ColdFusion and Java can work beautifully together. First, though, I must apologize for the ColdFusion presentation layer shown here. Due to space considerations, my presentation code is going to be woefully simple, but I hope it will show how easily ColdFusion can work with Java.

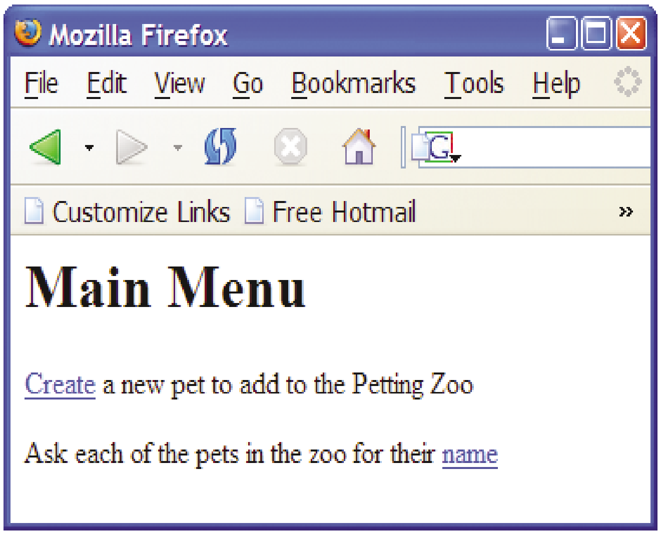
First, let's look at the Java code. I've used several of Java's features that ColdFusion lacks to create a more robust "domain model." (A domain model is a scale model code representation of the "domain" under study.) I start with a Pet interface. In Java, an interface provides the specification for a data type (including methods), but provides no implementation. Here's the code:

```
package pettingZoo;

public interface Pet {
   public String getName();
}
```

Now any class that wishes to can declare itself to be of type Pet. The requirement, set forth in the Pet interface, is that all "implementing classes" must implement a getName method that returns a string. The Java compiler will ensure that all implementing classes will be well behaved.

Next, I'll create an abstract class, Animal, that won't be used to create actual Animal objects, but is meant to be used by subclasses, where it will provide a base of inherited code:

```
package pettingZoo;

public abstract class Animal {
   private String food = null;

   public Animal(String food){
         setFood(food);
   }

   public String eat(){
         return "Thanks for the " + getFood();
   }

   public String getFood(){
         return this.food;
   }
   private void setFood(String food){
         this.food = food;
   }
}
```


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Customize Links Free Hotmail

Main Menu

Create a new pet to add to the Petting Zoo

Ask each of the pets in the zoo for their name

Figure 1: Main ColdFusion menu

Two more Java classes, Dog and Cat, will extend Animal while implementing the Pet interface by providing a getName method:

```
package pettingZoo;

public class Dog extends Animal implements Pet {
   private String name = null;

   public Dog(String name){
         super("kibbles and bits");
         setName(name);
   }

   public String getName() {
         return this.name;
   }
   private void setName(String name){
         this.name = name;
   }
}
```

```
package pettingZoo;

public class Cat extends Animal implements Pet{
   private String name = null;

   public Cat(String name){
         super("fish nibblies");
         setName(name);
   }

   public String getName(){
         return this.name;
   }
   public void setName(String name){
         this.name = name;
   }
}
```

Next, we have a RoboDog class that, while not extending Animal, implements Pet:

```
package pettingZoo;

public class RoboDog implements Pet{
   private String name = null;

   public RoboDog(String name){
         setName(name);
   }

   public String getName(){
         return this.name;
   }
   private void setName(String name){
         this.name = name;
   }
}
```

Our last Java class, PettingZoo, provides a way of modeling a very simple petting zoo. It has a method that allows us to add a pet to the collection and another method to return all of the pets:

```
package pettingZoo;

import java.util.*;
```

"I was totally intimidated by Java, but I knew I had to learn it. Your class taught me what I honestly thought I couldn't be taught." - Sharon T

Java for ColdFusion Programmers?

Java for ColdFusion Programmers, the five-day Hal Helms, Inc. training class is designed for ColdFusion programmers; no previous Java experience is needed. You'll learn how to think in objects and program in Java.

For class information and registration, come to halhelms.com.

```
public class PettingZoo {
    private List pets = null;

    public PettingZoo(){
            setPets(new ArrayList());
    }

    public void addPet(Pet pet){
            getPets().add(pet);
    }

    public List getPets(){
            return this.pets;
    }
    private void setPets(List pets){
            this.pets = pets;
    }
}
```

"But wait!" you say. "I don't understand all that Java stuff."

Well, that's the whole point. You don't need to. You're going to make use of a domain model written in Java to produce ColdFusion applications. The Java classes don't do anything by themselves: they just wait to be called upon. All you need to do is understand the API (the Application Programming Interface) for the classes you'll be using. In simpler terms, you need only understand what methods are available to call and what these will return. It's the same process you use when calling a built-in ColdFusion function (though with a different syntax). Java has a wonderful tool called "Javadoc" that automatically produces the API documentation you'll need from the underlying Java code.

With the Java out of the way, we can get down to our ColdFusion application code. First, I created an Application.cfm file:

```
<!--- set up application framework --->
<cfapplication name="CFandJavaDemo" />

<!--- create an application-scoped Java class, Petting Zoo --->
<cfif NOT IsDefined('Application.pettingZoo')>
    <cfset Application.pettingZoo = CreateObject('java',
'pettingZoo.PettingZoo').init() />
</cfif>
```

We're creating an object called pettingZoo from the Java class, PettingZoo, and placing it in the application scope. Now, for a main menu from MainMenu.cfm:

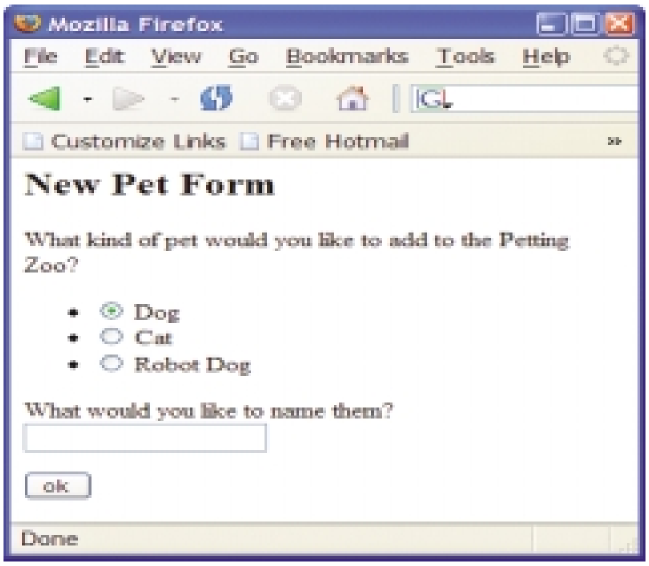
```
<h1>Main Menu</h1>

<p><a href="NewPetForm.cfm">Create</a> a new pet to add to the Petting
Zoo</p>

<p>Ask each of the pets in the zoo for their <a
href="PetGreeter.cfm">name</a></p>
```

It looks like that shown in Figure 1.

If the user elects to create a new pet, the NewPetForm page is displayed:

```
<h2>New Pet Form</h2>
<p>What kind of pet would you like to add to the Petting Zoo?
<form action="ProcessNewPetForm.cfm" method="post">
<ul>
    <li><input type="Radio" name="petClass" value="Dog" checked>
Dog</li>
    <li><input type="Radio" name="petClass" value="Cat"> Cat</li>
    <li><input type="Radio" name="petClass" value="RoboDog"> Robot
Dog</li>
</ul>
</p>
<p>What would you like to name them? <input type="Text"
name="petName"></p>
<p><input type="Submit" value="ok "></p>
</form>
```

The code produces the page shown in Figure 2: This forms asks the user the type of pet to be created and the new pet's name. When the form is submitted, ProcessNewPetForm is run:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Customize Links Free Hotmail

New Pet Form

What kind of pet would you like to add to the Petting Zoo?

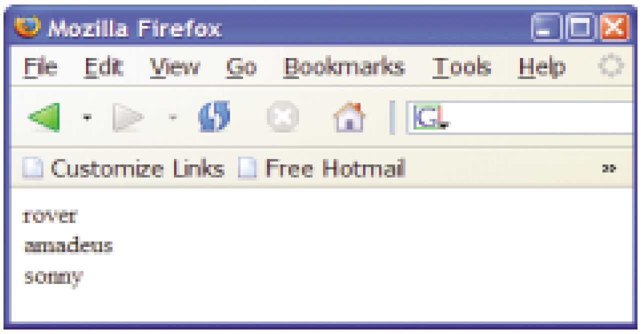
- [✓ Dog]
- Cat
- Robot Dog

What would you like to name them?

ok

Done

Figure 2: ColdFusion form for creating new Java pets


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Customize Links Free Hotmail

rover
amadeus
sonny

Figure 3: ColdFusion listing of the Java pets

```
<cfset newPet = CreateObject('java',
'pettingZoo.#form.petClass#').init(form.petName) />
<cfset Application.pettingZoo.addPet(newPet) />
<cflocation url="MainMenu.cfm" />
```

It's very short, letting Java do the heavy lifting of creating Pet objects and storing them in the petting zoo's collection. It then returns to the main menu.

If the user chooses to ask each of the pets for their name, PetGreeter is called:

```
<!--- Ask each pet in the pettingZoo for their name --->
<cfset pets = Application.pettingZoo.getPets() />
<cfloop from="1" to="#ArrayLen(pets)#" index="i">
    <cfoutput>
    #pets[i].getName()#<br />
    </cfoutput>
</cfloop>
```

PetGreeter loops over the pets returned by Java's PetZoo object, asking each pet – each Java Pet object, that is – for its name (see Figure 3).

With code understandable by any ColdFusion programmer, we've created a ColdFusion application that ties into a Java domain model.

The possibilities offered by the marriage of Java and ColdFusion are tremendous. It allows enterprises to call on the strength of each language and allows for the separation of the very different skills of server-side programming and presentation layer programming.

"When Blackstone is released, ColdFusion programmers will have an entirely new set of features and tools to work with"

When Blackstone is released, ColdFusion programmers will have an entirely new set of features and tools to work with, empowering them to produce richly interactive applications while integrating with Java enterprise domain models. It promises to be a lovely wedding – and we're all invited to the party.

About the Author

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

hal.helms@teamallaire.com

What's the Best Approach to Software Development?

An introduction to development methodologies

By Jeffry Houser

Developers each have their own approach to development. You may not know it, but you have one too. Having a set methodology allows you to sit back down at a project somewhere down the line and know what is going on immediately. Maintenance of the application is more cost effective. This article gives you an overview of the basic software development process, discusses what methodologies and frameworks are, and goes into some details on the methodology that I use in my own everyday development.

I have heard it said that the success of any software project occurs before a single line of code is ever written. Accordingly, we'll start our review by talking about the full software development life cycle. There are a few different models of software development, but they all contain similar elements, or phases.

These are the eight main phases:

1. ***Requirements:*** In the requirements phase, the client (software program user, your boss, whoever) defines some problem that they want to solve. You, as the programmer, will talk to the client about what they need a program to do. I like to think of the requirements phase as one where the client says, "This is what I want." In an ideal world, the client's request will be completely documented before I have to meet with them. Realistically, that rarely happens. This phase usually consists of multiple meetings with a client, project manager, or boss.
2. ***Specification:*** The specification phase comes after the requirements phase. Here we tell the client what we're going to do for them to solve their problem, as defined in the requirements phase. If we've done our job correctly, the requirements and specifications will sync up and all will be well in the world. I like to document both of these phases pretty heavily, and if possible have the client sign off on appropriate documentation.
3. ***Design:*** With the approval of a specification document, you can start your software design. You map out the flow of the program, mock up an interface (or rapid prototype), design the database, and flush out any business requirements not defined in the specifications.
4. ***Implementation:*** After you have the design all set, you can sit down and code. I'm sure that is what most of us do best. The implementation makes the design work.
5. ***Quality assurance:*** In this phase, you test the application from start to finish, looking for problems with the code or logic. Next to requirements, the client often has the most involvement in this phase. They will know immediately at this point if something is seriously wrong with the application.
6. ***Integration and rollout:*** In the past this would involve moving from machine to machine to upgrade to the new version of the software. With Web applications, those days are over. In the Web world, rollout is easy. Just tell people to point their browser to the new application.
7. ***Maintenance:*** Over time, businesses change – and so do requirements. The application will need to change to meet those changing business requirements. Often in the maintenance phase you go through the whole software development process in a smaller way, specifying requirements for changes, defining the specifications, modifying the design, and so on.
8. ***Retirement:*** All good things must come to an end. In this phase we put the application to rest. Business requirements have changed so much over the years that it's more cost effective to replace the application than to modify it.

As you might have guessed, the phases often overlap. Often requirements and specifications are lumped into a single phase, for example. And some design work may be done during specifications; or during the implementation. Debugging is done during both the implementation and testing phases. Having the client involved in all phases of the project is often critical for its success.

There are different models of software development using these phases. The waterfall model is the most common. It moves from one phase to the other in tandem. The rapid prototype model starts by building a rapid prototype, or mockup, of the application. A specification is built from the rapid prototype, and the rest of the phases continue one after the other. The rapid prototype or waterfall models encompass full applications.

There's a third model called the iterative model, that follows the whole process for each individual application feature. The intent of this model is that release cycles will come every few weeks. This model works really well during large projects. You wouldn't want to be halfway through development on a yearlong project and then find out that everything you've done is wrong. I often like to use this model for the maintenance phase also.

My Development Methodology

The rest of this article will specifically talk about how I myself develop during the implementation phase of a project. I'm going to explain my method for organizing code and approaching development. Every template I write starts with a documentation header. Good documentation is often crucial when modifying code at some future point.

First, I include a description of the template. Why am I creating this template, what code does it contain? I'll also include the name of the file and the date it was created. When working as part of a team, I'll specify the name of the creator (me), and my company name, if I'm developing for a client.

Second, I'll specify where the code fits into the flow of logic. Is it an include file, a custom tag, a form page, or something else? If there is a multistep process, what step does this file represent? What file represents the previous step? What file represents the next step?

Next I'll document any file dependencies such as custom tags, include files, or ColdFusion components that need to exist in order for this template to run correctly. Then I'll document variable dependencies. What variables must be defined for this template to properly process? Do we need any URL variables? Form variables? Are they required or optional? If they are optional what are their default values?

Sometimes it may seem like a lot of work to create this documentation up front, but you'll thank yourself down the line. It takes minimal extra effort when you create the page and can save loads of time during the maintenance phase.

Documentation aside, I try to keep each template as modular as possible. That means that each template does only a single set of related actions. Here are the common types of templates you'll encounter during your development:

- ***Application.cfm & OnRequestEnd.cfm:*** The Application.cfm is a page that ColdFusion runs at the beginning of every request. I use this page to set up the basic application, directory locations variables, and datasource names. OnRequestEnd.cfm runs at the end of every request. It is not used often.
- ***Header and footer:*** I usually have only one header and footer file in a site, called header.cfm and footer.cfm respectively. The header contains the standard navigation and most common graphical elements of the site. The footer contains any disclaimers at the bottom of every page. I use a cfinclude to put these two templates into every other template. Between the header and footer goes the individual content. I don't put the header information in the Application.cfm and footer information in OnRequestEnd.cfm is because I like to reserve those files for processing code, not display code.
- ***Index:*** The index template is your home page. Some Web servers are programmed to look for default.cfm instead of index.cfm, but many will look for both.
- ***List page:*** A list page is a page that lists data based on certain criteria. For example, if you are creating a page that lists all the users who have registered on the site, you might name this page register.cfm.
- ***Input***: Input pages are pages that accept input from the user. In standard Web development an input page will contain an HTML form. I like to distinguish an input page by putting an "i" at the end of the file name. A file that accepts input for a user registration would be registeri.cfm.
- ***Processing:*** Processing pages are those that a form submits onto. These pages usually verify the data, and then insert or update it to the database. I like to distinguish these with a "p" at the end

of the file name. In our registration example, the registration-processing page would be registerip.cfm. That signifies that we are processing the registration input.

- ***Update:*** Pages that update data are very similar to input pages, except the forms are already filled with data. I like to distinguish these files with a "u". A registration update page would be registeru.cfm. The processing page for the update registration functionality would be registerup.cfm.
- ***Verification:*** Verification pages are pages that lie between input and processing. They display all the data to the user and give the user an option of changing data before it goes into the database. If you want to have the user verify their data before committing it to the database, use a verification page.
- ***Delete:*** A delete page is one for deleting data. I distinguish these files with a "d" character. For example, to delete a registration, the file would be named registerd.cfm. Often these pages will trigger a flag to make the data inactive instead of completely deleting the data.

I've worked on sites that have tried to put a lot of the functionality into a single template, such as create, edit, and delete. I've had much better luck maintaining templates that don't do everything in a single template. There's also less processing because you don't have to figure out which "actions' of the template you're supposed to be executing. Using this method can make code less portable than having everything in a single template. If consistent naming conventions are used, this is rarely a problem.

Since the release of ColdFusion MX, I have been moving most of the functional code inside CFCs. For example, the verification page will call methods, which verify the data, but won't actually verify the data. If an error is returned, it displays the appropriate error. The process page will call a method to commit the data to the database. I've been storing the CFCs in the session scope, so I don't have to worry about excessive parameter passing from one template to another. The input page collects the data, the verification page stores it in component properties, and the processing page commits it to the database.

Other Development Methodologies and Frameworks

There's often a lot of confusion as to what a methodology is, what a framework is, and what the differences between them are.

A methodology is an organized approach to writing and organizing your code. What I've defined in this document is a rough methodology that I use for code development. A framework is a collection of code used as a template for writing applications. There is often a tradeoff for using a framework in your development and that is that it will most likely contain code you don't need to use. The performance hindrance of unused code is offset by the decrease in a development schedule. Using a framework will also make your code more consistent. In performance critical -applications you probably won't want to build your application within a framework.

Here are some common methodologies and frameworks used throughout the ColdFusion world:

- ***Fusebox:*** Fusebox is one of the most popular methodologies out there. Fusebox 1 was a strict methodology; although current versions provide some semblance of a framework. The most current version is Fusebox 4 and more information can be found at www.fusebox.org. Anyone looking at Fusebox should also take a look at the Fusebox Lifecycle Process (FLiP), which handles the full software development life cycle.
- ***cfobjects:*** cfobjects is a framework used to apply object-oriented principles to ColdFusion development. It is developed to work with pre-CFMX versions and consists of a lot of custom tags, plus a toolbar for ColdFusion Studio (HomeSite +). More information can be found at http://cfobjects.sourceforge.net.
- ***Mach-II:*** Mach-II is a framework used to apply object-oriented design principles to ColdFusion from development. It's relatively new in the world of development and was spun off development that was originally supposed to be Fusebox MX. You can find more information about Mach-II at www.mach-ii.com/. Macromedia has developed some of their site using the Mach-II framework.

"Having the client involved in all phases of the project is often critical for its success"

Summary

This was intended as a high-level overview of approaches to development. No matter what type of development you are doing, you'll need a methodology – having one that you always follow keeps things consistent if nothing else. When working with teams, it's best to decide on a common methodology that all developers will use. Not every methodology will work in every situation. It's best to document the methodology that you are using as developers enter and leave your project.

About the Author

Jeffry Houser has been working with computers for over 20 years and in Web development for over 8 years. He owns a consulting company, and has authored three separate books on CF, most recently ColdFusion MX: The Complete Reference *(McGraw-Hill Osborne Media).*

jeff@instantcoldfusion.com

A new tool for MX professional developers and designers...

www.mxdevelopersjournal.com 0.9

MX developer's journal

Toppling the...

Tower of Babel with MX

How to produce the intense application:

BabelChat

ADVERTISE
Contact: Robyn Forma
robyn@sys-con.com
(201) 802-3022
for details on rates
and programs

SUBSCRIBE
www.sys-con.com/
mx/subscription.cfm
1(888) 303-5282

SYS-CON MEDIA

MX developer's journal

— continued from page 7

This is what I love about lists and about ColdFusion – the enthusiasm and fun people can have. I finished off the thread by showing him yet another way to accomplish the same thing which I thought he'd have fun playing with:

```
<cfscript>
 function showStruct(stVal){
 var i = "";
 for (i in arguments.stVal){
  writeOutput("The value of the " & i & " key
is " & arguments.stVal[i] & "<br />");
 }
 return;
 }
 writeOutput("<h3>The Form Scope</h3>");
 showStruct(form);
 writeOutput("<h3>The URL Scope</h3>");
 showStruct(url); writeOutput("<h3>The
Variables Scope</h3>");
 showStruct(variables);
</cfscript>
```

Accessing Dynamically Named Variables

This particular thread didn't ever have to delve into the ins and outs of accessing dynamically named variables, but I frequently see posts enquiring about techniques to do this. The trick is to become comfortable with using array-notation to access the values in an object (or structure). Many people initially try to do this with the evaluate() function, which does work but isn't as elegant from either a readability or a performance point of view (plus Ray Camden will publicly flog you for it).

I'm not going to get into the details of this – you can see the recent "CF-101" article written by Jeffry Houser for details [*CFDJ*, Vol. 6 Issue, 2]. But I will show a relatively simple example of how to do this, for the benefit of readers that are new to dynamically named variables. I'll use Mike's original scenario of having a form with checkboxes named "size1" to "size10".

In order to create the form fields you have to remember that form fields are nothing but text rendered by the browser. You can name them dynamically the same way you'd output any other dynamic text to screen and do something simple like the code shown in Listing 1.

That's the easy part – the form submits to itself, so now I add an "if" statement to the top of the page and in the event that the form was submitted, I'm just going to output the names of the checkboxes that were checked when the form was submitted, as well as their corresponding values. There are several ways to do this – here's one:

```
<cfscript>
 if (structKeyExists(form, "submit")){
 writeOutput("You submitted:<br />");
  for (i=1; i lte 10; i = i + 1){
    thisField = "size" & variables.i;
    if (structKeyExists(form,
variables.thisField)){
    writeOutput("Size " & variables.i & "
(value = " & form[variables.thisField] & ")<br
/>");
    }
    }
 }
</cfscript>
```

The same thing using CF tags rather than CFSCRIPT would look like this:

```
<cfif structKeyExists(form, "submit")>
 You submitted:<br />
 <cfoutput>
 <cfloop from="1" to="10" index="i">
  <cfset thisField = "size" & variables.i>
    <cfif structKeyExists(form,
variables.thisField)>
    Size #variables.i# (value = #form[vari-
ables.thisField]#)<br />
  </cfif>
 </cfloop>
 </cfoutput>
</cfif>
```

The ability to dynamically create and name form fields is very powerful when you're creating sites and applications that display and collect dynamic content. CFML offers several (syntax) techniques that make it easy not only to dynamically gather but also to dynamically access form information. The techniques described in this article are useful in survey and questionnaire applications, content management systems, and a variety of other applications.

It's important to remember that a variety of ways to code the same functionality almost always exists and that you should experiment and become comfortable with them all. Variety, after all, is the spice of life.

"This is what I love about lists and about ColdFusion – the enthusiasm and fun people can have"

Listing 1:

```
<cfoutput>
 <form action="#cgi.script_name#" method="post">
 <table>
  <cfloop from="1" to="10" index="i">
  <tr>
   <td><strong>Size #variables.i#:</strong></td>
   <td><input type="checkbox" name="size#variables.i#"
value="#variables.i#" /></td>
  </tr>
  </cfloop>
  <tr>
   <td><input type="submit" name="submit" value="Submit Form" /></td>
   <td><input type="reset" value="Reset Form" /></td>
  </tr>
 </table>
 </form>
</cfoutput>
```

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

ATTN:

STEP UP to the mike and be...

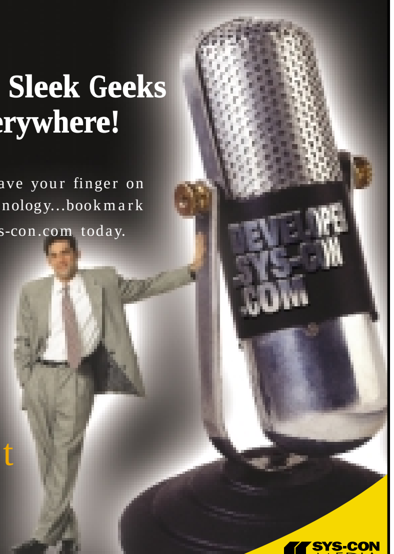
Go to http://developer.sys-con.com

HEARD!

Calling Sleek Geeks Everywhere!

Make sure you have your finger on the pulse of i-Technology...bookmark http://developer.sys-con.com today.

i-Technology News
i-Technology Views
i-Technology Comment
i-Technology Debate



SYS-CON MEDIA

CFDJ Advertiser Index

ADVERTISER	URL	PHONE	PAGE
FLASHFORWARD	WWW.FLASHFORWARDCONFERENCE.COM	877-435-2744	27
ACTIVEPDF	WWW.ACTIVEPDF.COM		4
CFDJ	WWW.SYS-CON.COM/COLDFUSION/	888-3035282	49
CFDYNAMICS	WWW.CFDYNAMICS.COM	866.233.9626	23
CFUN	WWW.CFCONF.ORG/CFUN-04/	301-424-3903	COVER III
CLEARNOVA	WWW.CLEARNOVA.COM/THINKCAP		17
CF RESOURCE CD	WWW.SYS-CON.COM/FREECD	888-303-5282	47
EDGE WEB HOSTING	WWW.EDGEWEBHOSTING.NET	866-334-3932	15
HAL HELMS, INC	WWW.HALHELMS.COM		29
HOSTMYSITE.COM	WWW.HOSTMYSITE.COM/CFDJ	877-248-4678	25
IS&SECURITY	WWW.ISSJOURNAL.COM	888-303-5282	33
INTERAKT ONLINE	WWW.INTERAKTONLINE.COM		21
INTERMEDIA.NET	WWW.INTERMEDIA.NET	800.379.7729	COVER IV
IT SOLUTIONS GUIDE	WWW.SYS-CON.COM	201-802-3021	43
JAVA RESOURCE CD	WWW.SYS-CON.COM/FREECD	888-303-5282	42
MACROMEDIA	WWW.MACROMEDIA.COM/GO/DWUPDATED		COVER II
MACROMEDIA	WWW.MACROMEDIA.COM/GO/DWUPDATED		3
MXDJ	WWW.SYS-CON.COM/MX/SUBSCRIPTION.CFM	888-303-5282	35
PAPERTHIN	WWW.PAPERTHIN.COM	800.940.3087	11
SYS-CON ENEWSLETTER	WWW.SYS-CON.COM	888-303-5282	49
SYS-CON MEDIA	HTTP://DEVELOPER.SYS-CON.COM	888-303-5282	39
SYS-CON PUBLICATION	WWW.SYS-CON.COM		31
SYS-CON REPRINTS	KRISTIN@SYS-CON.COM	201-802-3026	41
WEB SERVICES EDGE EAST	WWW.SYS-CON.COM/EDGE	201-802-3045	39
WINMILL SOFTWARE	WWW.WINMILL.COM	888-711-6455	6

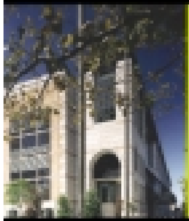
web services EDGE conference&expo

Web Services Edge 2005 East

International Web Services Conference & Expo

Announcing Web Services Edge 2005 Call for Papers Now Open!

Submit your proposal by August 15, 2004



The Largest i-Technology Event of the Year!

Guaranteed Minimum Attendance 3,000 Delegates

Hynes Convention Center, Boston, MA

February 15-17, 2005

Join us in delivering the latest, freshest, and most proven Web service solutions at the Fifth Annual Web Services Edge 2005 East-International Conference & Expo as we bring together IT professionals, developers, policy makers, industry leaders and academics to share information and exchange ideas on technology trends and best practices in secure Web services and related topics including:

- Transitioning Successfully to SOA
- Federated Web services
- ebXML
- Orchestration
- Discovery
- The Business Case for SOA
- Interop & Standards
- Web Services Management
- Messaging Buses and SOA
- Enterprise Service Buses
- SOBAs (Service-Oriented Business Apps)
- Delivering ROI with SOA
- Java Web Services
- XML Web Services
- Security
- Professional Open Source
- Systems Integration
- Sarbanes-Oxley
- Grid Computing
- Business Process Management
- Web Services Choreography

Interested in exhibiting, sponsoring or advertising?

Becoming a Web Services Edge exhibitor or sponsor offers you a unique opportunity to best position your company's message and visibility to a targeted audience of Web services professionals. Through a wide variety of pre-show, on-site, and post-show marketing programs and opportunities, Web Services Edge 2005 East will serve industry stalwarts as well as exciting start-up companies in reaching the world's most mature internet-based regional market, the US Northeast Region, at Boston's best-known IT business address, the Hynes Convention Center. For Exhibit & Sponsorship information please contact: Grisha Davida, 201-802-3004, e-mail: grisha@sys-con.com

Tuesday, 2/15: Conference & Expo | Wednesday, 2/16: Conference & Expo | Thursday, 2/17: Conference & Expo

Sponsored by:

JDJ WebServices .NET Journal WebSphere Journal Information Storage + Security Journal XML Journal MX Developer's Journal wldj

Contact for Conference Information: Lin Goetz, 201-802-3045, lin@sys-con.com

www.sys-con.com/edge

PRODUCED BY SYS-CON EVENTS

ColdFusion Components and Data Abstraction

CFCs provide basic object functionality to CF developers

PART 2

By Ben Forta

In my last column we looked at using ColdFusion Components to abstract database access, essentially divorcing presentation code from anything database specific. As you will recall, the benefit of this was that when a database change occurred (a column being renamed, for example), presentation code was not impacted at all. In this column I'll take this concept one step further.

Beyond Simple Abstraction

First, let's review. Here is simple code to retrieve a list of employees:

```
<!--- Get employees --->
<CFINVOKE COMPONENT="emps"
          METHOD="list"
          RETURNVARIABLE="employees">

<!--- Display list --->
<UL>
 <CFOUTPUT QUERY="employees">
  <LI>#LastName#, #FirstName#</LI>
 </CFOUTPUT>
</UL>
```

<CFINVOKE> invokes list in component emps, which obtains an employee list. emps.cfc contains (as we left it last month), with two methods: list is used to obtain a list of users, and update is used to update an employee's name (see Listing 1). So far so good.

Components Are NOT Query Replacements

Thus far we've used CFCs and methods as simple query replacements. That's not a bad place to start, and if you were to never again use a <CFQUERY> in your .cfm files, that would be a very good thing indeed.

But CFCs aren't designed to simply be inline query replacements. In fact, CFCs need not map to database tables or queries at all. For example, if you had an online store you'd probably work with several types of components:

- A customer component to provide access to customer specifics and any customer related processing
- A catalog component used for catalog lists and searches
- An item component used to obtain item specifics, as well as pricing, availability, images, and more
- An order component
- and so on

These components may map to tables. Internally, a customer may indeed be a single row in a database table. But then again, it may not. The truth is that internal storage doesn't really matter. What's more important is that components are logical entities used to encapsulate and abstract related functionality. In more understandable terms, components are like little black boxes that contain everything you need to perform specific operations for specific functions.

Components As Objects

This brings us to the subject of using components as objects. The best way to understand this concept is with another example. Look at the following code:

```
<!--- Get employees object --->
<CFOBJECT COMPONENT="emps"
          NAME="emp">

<!--- Get employees --->
<CFINVOKE COMPONENT="#emp#"
          METHOD="list"
          RETURNVARIABLE="employees">

<!--- Display list --->
<UL>
 <CFOUTPUT QUERY="employees">
  <LI>#LastName#, #FirstName#</LI>
 </CFOUTPUT>
</UL>
```

If you were to run this code, it would generate the exact same output as the previous invocation example. But something is very different here.

Previously, <CFINVOKE> was used to access a component and then invoke a method. It did both of those tasks, all in one tag. Here we have separated the component access from the method invocation. <CFOBJECT> loads the component as an object – the new object is named emp. This <CFOBJECT> call does not execute any method; in fact, no method name is even specified. It is <CFINVOKE> that invokes the list method in the already loaded component object. (Notice that #'s are used in the COMPONENT attribute in <CFINVOKE. This is because it refers to a variable as opposed to a file name).

The advantage of this type of access and invocation is that it allows for components to be instantiated (obtain an instance of) independent of any method invocation. This allows for multiple invocations against the same component (perhaps a series of inserts or updates). It also allows components to persist (to stick around, especially if the object is in a persistent scope like SESSION or APPLICATION). And it allows for alternate forms of invocation. Look at this example:

```
<!--- Get employees object --->
<CFOBJECT COMPONENT="emps"
          NAME="emps">

<!--- Get employees --->
<CFSET employees=emps.list()>

<!--- Display list --->
<UL>
 <CFOUTPUT QUERY="employees">
  <LI>#LastName#, #FirstName#</LI>
 </CFOUTPUT>
</UL>
```

Once again, the code here accomplishes the same result as in the previous examples, but notice that there's no <CFINVOKE> tag used here. Rather, a simple <CFSET> is used to save the results of *emp.list()* (the *list* method in component *emp*) to a variable named *employees*.

This form of invocation is used when working with objects, which is what the component here is being used as. Once a component is loaded as an object, it can be used in lots of different ways, and repeatedly too.

Methods, More Methods, and Even More Methods

Now you know how to use components as objects. Great. Now what? The next step is to define all sorts of methods in your component, methods for anything you may need to know about or do.

For example, imagine we had a second component we could use. Unlike *emps.cfc* which is used to access all employees (lists of employees), *emp.cfc* provides access to a single employee, and everything that you'd need to know about him or her.

Look at this simple invocation in Listing 2. <CFOBJECT> instantiates the employee object, just as we saw previously. But which employee's object is this? Actually, it's not associated with any employee – we'd need to do that ourselves. The <CFSET> statement invokes an *Init* method (as in *initialize*) and passes an employee ID as an argument (we've hard-coded it to 22 here, but you could pass a variable or any expression instead).

Once initialized, the employee object can provide access to everything you'd want to know about an employee. The *GetName* method (invoked as *#emp.GetName()#*) returns the employee name, GetStartDate returns the employee start date (which is formatted using the DateFormat() function), *GetYears* returns the number of years at the company, and *EligibleForCar* returns *true* if the employee has worked long enough to be entitled to a company car (hey, we can all dream!). Load the object once, initialize it, and then use it over and over as needed.

All of the code is presentation code. There is no database access, there is no logic, no calculations (as would be needed to figure our company car eligibility). All of that is buried in the little black box, the component. And once that component has been instantiated and initialized, all of the other methods can be invoked as needed.

So what does the *emp.cfc* component look like? And how does it *remember* the employee that it is associated with? The code is shown at Listing 3.

Component *emp.cfc* contains lots of methods. *Init* is an important one; it requires that an employee ID be passed to it as an argument, and it then uses it to query a database for employee details (using the *emp_id* in the SQL WHERE clause). The query name is *THIS.emp*, *THIS* is a special scope that refers to the CFC itself. As the query is placed into *THIS*, it persists for as long as the query does, and will be usable on subsequent method invocations. This is why, after calling init in the calling page, the code was able to make all of those subsequent method invocations.

The *GetName* method returns an employee name, extracted from the saved query. The CFC need not query the database again, as the query created in *init* still exists (in *THIS*).

GetFirstName and *GetLastName* do exactly what their names suggest. They are not used in our example, but you will generally want to create all the methods that you may end up using at some point. There's no downside to doing this, and the extra effort up front will make the components far more reusable.

GetStartDate returns the employee start date stored in the table. *GetYears* uses the start date to calculate the numbers of years of employment. This is a great example of the type of calculation that's too often erroneously placed in presentation code (where it's used, but where it absolutely does not belong).

The *EligibleForCar* method is similar; it does the calculation (need to have been employed for at least 5 years) and simply returns *true* or *false*. This type of calculation does not belong in presentation code, ever. Logic, business rules, data abstraction, all of that and more belong in CFC code. Presentation code is for, well, presentation.

We're Just Getting Started

We've just scratched the surface. CFCs are the most important enhancement to the CFML language since the language was first created almost a decade ago. At a minimum, CFCs should subtract all database access. I'll say it again, no more <CFQUERY> tags in your presentation code.

But beyond simple database abstraction, ColdFusion Components should be used for all data abstraction, including calculations, business logic, any data processing, and more.

Yes, it takes a little more time up front. But once you try it you'll be pleasantly surprised to discover that it really is just a little more time.

And the upside?

- More reusable code.
- More manageable code.
- Performance gains (when using persistence properly).
- Dramatically reduced risks when making changes.
- The ability to easily replace or add presentation layers. (Need a Flash front end? Flex? Web services? Those are all alternate presentation layers against the same back-end CFC).

In other words, there's plenty to gain, and nothing to lose.

Summary

ColdFusion Components, first introduced in ColdFusion MX, provide the fundamental building blocks used to design applications the right way. They provide some of the power of objects while retaining the simplicity that is uniquely ColdFusion. In the last column we looked at basic data abstraction, separating data (and data integration) from presentation. This time we looked at encapsulating more than just data processing, persistence, and more. For your next project, you'll most decidedly want to use ColdFusion Components.

About the Author

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

ben@forta.com

Listing 1

```
<CFCOMPONENT>

<!--- List employees --->
<CFFUNCTION NAME="List"
             RETURNTYPE="query"
             OUTPUT="false">
 <!--- Get data --->
 <CFQUERY NAME="employees"
          DATASOURCE="CompanyInfo">
 SELECT Emp_ID, LastName, FirstName
 FROM Employee
 ORDER BY LastName, FirstName
 </CFQUERY>
 <!--- Return data --->
 <CFRETURN employees>
</CFFUNCTION>

<!--- Update an employee --->
<CFFUNCTION NAME="Update"
             RETURNTYPE="boolean"
             OUTPUT="false">
 <!--- Arguments --->
 <CFARGUMENT NAME="Emp_ID"
             TYPE="numeric"
             REQUIRED="yes">
 <CFARGUMENT NAME="LastName"
             TYPE="string"
             REQUIRED="yes">
 <CFARGUMENT NAME="FirstName"
             TYPE="string"
             REQUIRED="yes">

 <!--- The update --->
 <CFQUERY DATASOURCE="CompanyInfo">
 UPDATE Employee
 SET LastName = '#ARGUMENTS.LastName#',
     FirstName = '#ARGUMENTS.FirstName#'
 WHERE Emp_ID = #ARGUMENTS.Emp_ID#
 </CFQUERY>
 <CFRETURN true>
</CFFUNCTION>
```

```
</CFCOMPONENT>
```

Listing 2

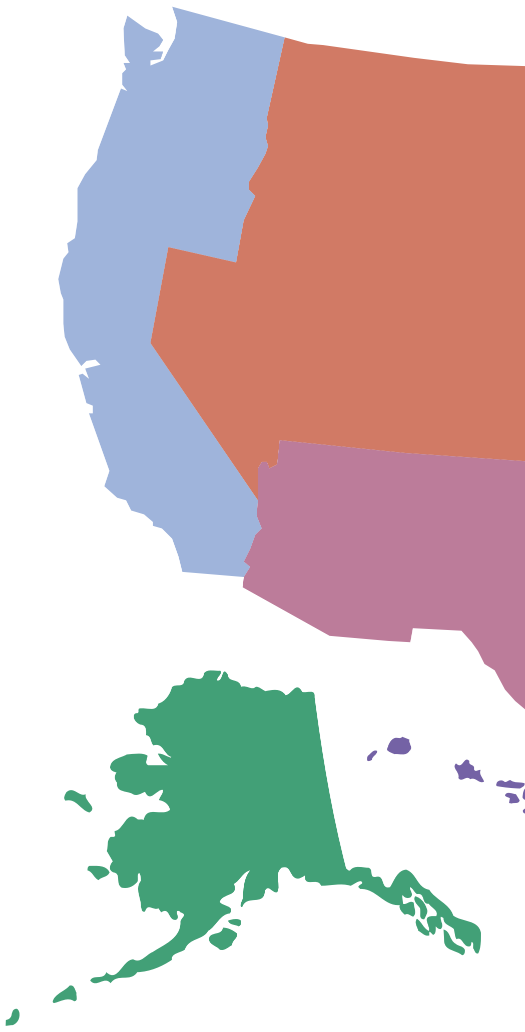
```
<!--- Get employee object --->
<CFOBJECT COMPONENT="emp" NAME="emp">

<!--- Initialize employee --->
<CFSET emp.Init(22)>

<!--- Display details --->
<CFOUTPUT>
Name:
#emp.GetName()#<BR>
Start Date:
#DateFormat(emp.GetStartDate())#<BR>
Years With Us:
#emp.GetYears()#<BR>
Company Car:
#YesNoFormat(emp.EligibleForCar())#<BR>
</CFOUTPUT>

<CFCOMPONENT>

<!--- Initialize --->
<CFFUNCTION NAME="Init"
            RETURNTYPE="boolean"
            OUTPUT="no">
 <CFARGUMENT NAME="emp_id"
             TYPE="numeric"
             REQUIRED="yes">
  <!--- Get employee info --->
  <CFQUERY NAME="THIS.emp"
           DATASOURCE="CompanyInfo">
  SELECT Emp_ID, LastName, FirstName,
               StartDate, Dept_Name
  FROM Employee, Departmt
  WHERE Employee.Dept_ID = Departmt.Dept_ID
   AND emp_id = #ARGUMENTS.emp_id#
  </CFQUERY>
 <CFRETURN true>
</CFFUNCTION>

<!--- Get employee name --->
<CFFUNCTION NAME="GetName"
            OUTPUT="no"
            RETURNTYPE="string">
 <CFRETURN "#THIS.emp.LastName#,
            #THIS.emp.FirstName#">
</CFFUNCTION>

<!--- Get employee first name --->
<CFFUNCTION NAME="GetFirstName"
            OUTPUT="no"
            RETURNTYPE="string">
 <CFRETURN THIS.emp.FirstName>
</CFFUNCTION>

<!--- Get employee last name --->
<CFFUNCTION NAME="GetLastName"
            OUTPUT="no"
            RETURNTYPE="string">
 <CFRETURN THIS.emp.LastName>
</CFFUNCTION>

<!--- Get employee start date --->
<CFFUNCTION NAME="GetStartDate"
            OUTPUT="no"
            RETURNTYPE="date">
 <CFRETURN THIS.emp.StartDate>
</CFFUNCTION>

<!--- Get years with us --->
<CFFUNCTION NAME="GetYears"
            OUTPUT="no"
            RETURNTYPE="numeric">
 <CFRETURN DateDiff("yyyy",
                    THIS.emp.StartDate,
                    Now())>
</CFFUNCTION>

<!--- Determine if eligible for car --->
<CFFUNCTION NAME="EligibleForCar"
            OUTPUT="no"
            RETURNTYPE="numeric">
 <CFSET VAR eligible=FALSE>
 <CFIF GetYears() GTE 5>
  <CFSET eligible=TRUE>
 </CFIF>
 <CFRETURN eligible>
</CFFUNCTION>

</CFCOMPONENT>
```

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

ColdFusion

For more information go to…

U.S.



Alabama
Huntsville
Huntsville, AL CFUG
www.nacfug.com

Alaska
Anchorage
Alaska Macromedia User Group
www.akmmug.org

California
San Francisco
Bay Area CFUG
www.bacfug.net

California
Riverside
Inland Empire CFUG
www.sccfug.org

California
EL Segundo
Los Amgeles CFUG
www.sccfug.org

California
Irvine
Orange County CFUG
www.sccfug.org

California
Davis
Sacramento, CA CFUG
www.saccfug.org

California
San Jose (temporary)
Silicon Valley CFUG
www.siliconvalleycfug.com

California
San Diego
San Diego, CA CFUG
www.sdcfug.org/

California
Long Beach
Southern California CFUG
www.sccfug.org

Colorado
Denver
Denver CFUG
www.denvercfug.org/

Delaware
Kennett Square
Wilmington CFUG
www.bvcfug.org/

Delaware
Laurel
Delmarva CFUG
www.delmarva-cfug.org

Florida
Jacksonville
Jacksonville, FL CFUG
www.jaxfusion.org/

Florida
Winter Springs
Gainesville, FL CFUG
www.gisfusion.com/

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

Florida
Tallahassee
Tallahassee, FL CFUG
www.tcfug.com/

Florida
Palm Harbor
Tampa, FL CFUG
www.tbmmug.org

Georgia
Atlanta
Atlanta, GA CFUG
www.acfug.org

Illinois
East Central
East Central Illinois CFUG
www.ecicfug.org/

Indiana
Avon
Indianapolis, IN CFUG
www.hoosierfusion.com

Indiana
Mishawaka
Northern Indiana CFUG
www.ninmug.org

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

Kentucky
Louisville
Louisville, KY CFUG
www.kymug.com/

Louisiana
Lafayette
Lafayette, LA MMUG
www.cflib.org/acadiana/

Maryland
Lexington Park
California, MD CFUG
http://www.smdcfug.org

Maryland
Rockville
Maryland CFUG
www.cfug-md.org

Massachusetts
Quincy
Boston, MA CFUG
www.bostoncfug.com

Michigan
Portlalnd
Mid Michigan CFUG
www.coldfusion.org/pages/index.cfm

Minnesota
Brooklyn Park
Twin Cities CFUG
www.colderfusion.com

Missouri
Overland Park
Kansas City, MO CFUG
www.kcfusion.org

Missouri
O'Fallon
St. Louis, MO CFUG
www.stlmmug.com/

New Jersey
Princeton
Central New Jersey CFUG
http://www.cjcfug.us/

Nevada
Las Vegas
Las Vegas CFUG
www.sncfug.com/

New York
Albany
Albany, NY CFUG
www.anycfug.org

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

New York
Syracuse
Syracuse, NY CFUG
www.cfugcny.org

North Carolina
Raleigh
Raleigh, NC CFUG
www.ccfug.org

Ohio
Dayton
Greater Dayton CFUG
www.cfdayton.com

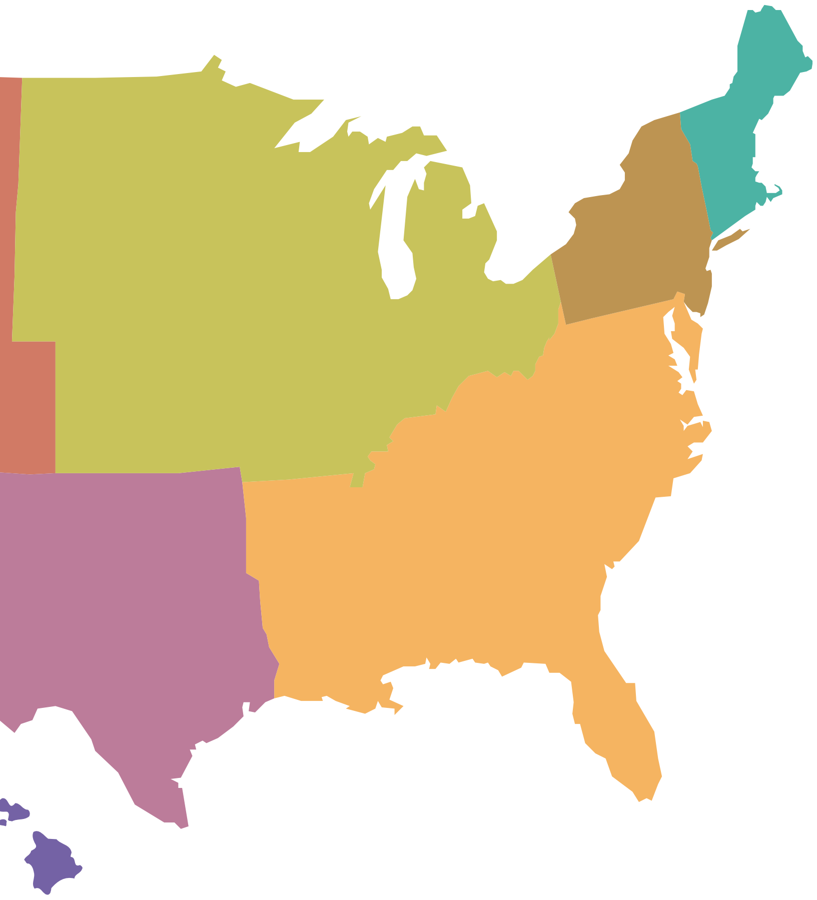
Oregon
Portland
Portland, OR CFUG
www.pdxcfug.org

Pennsylvania
Carlisle
Central Penn CFUG
www.centralpenncfug.org

Pennsylvania
Exton
Philadelphia, PA CFUG
www.phillycfug.org/

User Groups

http://www.macromedia.com/cfusion/usergroups



Pennsylvania
State College
State College, PA CFUG
www.mmug-sc.org/

Rhode Island
Providence
Providence, RI CFUG
www.ricfug.com/www/meetings.cfm

Tennessee
LaVergne
Nashville, TN CFUG
www.ncfug.com

Tennessee
Germantown
Memphis, TN CFUG
www.mmug.mind-over-data.com

Texas
Austin
Austin, TX CFUG
www.cftexas.net/

Texas
Corinth
Dallas, TX CFUG
www.dfwcfug.org/

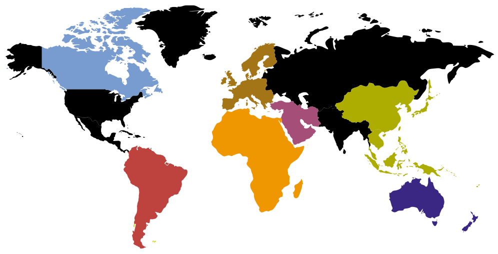
Texas
Houston
Houston Area CFUG
www.houcfug.org

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

About CFUGs

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

INTERNATIONAL



Australia
ACT CFUG
www.actcfug.com

Australia
Queensland CFUG
www.qld.cfug.org.au/

Australia
Southern Australia CFUG
www.cfug.org.au/

Australia
Victoria CFUG
www.cfcentral.com.au

Australia
Western Australia CFUG
www.cfugwa.com/

Brazil
Brasilia CFUG
www.cfugdf.com.br

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

Brazil
Sao Paulo CFUG
www.cfugsp.com.br

Canada
Kingston, ON CFUG
www.kcfug.org

Canada
Toronto, ON CFUG
www.cfugtoronto.org

Ireland
Dublin, Ireland CFUG
www.mmug-dublin.com/

Italy
Italy CFUG
www.cfmentor.com

Japan
Japan CFUG
cfusion.itfrontier.co.jp/jcfug/jcfug.cfm

Scotland
Scottish CFUG
www.scottishcfug.com

South Africa
Joe-Burg, South Africa CFUG
www.mmug.co.za

South Africa
Cape Town, South Africa CFUG
www.mmug.co.za

Spain
Spanish CFUG
www.cfugspain.org

Switzerland
Swiss CFUG
www.swisscfug.org

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

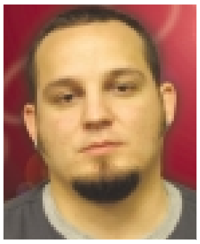
Turkey
Turkey CFUG
www.cftr.net

United Kingdom
UK CFUG
www.ukcfug.org

Creating Free PDFs from Your CF Application

Not only functional, but presentable too!

PART 2



By Nate Nelson

Here's how to put your knowledge of HTML to work by creating free PDFs with FOP (Formatting Objects Processor).

In the May issue of ***CFDJ***, I covered the basics of utilizing FOP from Apache to dynamically create free PDFs in your ColdFusion application. If you enjoyed that article you will certainly enjoy this one as we dive deeper into the capabilities of this process. After reading both articles you should be able to not only generate free PDFs but also to display presentable results simply by using the HTML that you already know. In this article I will demonstrate how to do this by further defining XSLT and XSL-FO.

A Quick Recap

In part 1 of this article, available at www.sys-con.com/story/?storyid=44771&DE=1, I covered everything you need to know in order to create your first free dynamic PDF using CF, including an introduction to FOP, how to generate and transform XML, and how to invoke FOP.

What's Next?

If you read part 1 of this article, you may be thinking something like "Okay, this is pretty easy.... I can create a PDF in my ColdFusion app with no sweat, but now I want to make it look nice and add some formatting to it." After all, if we were just spitting out text all the time it wouldn't even be necessary to use a PDF file to display your results. In this article I'm going to go back to further define a couple of topics that I briefly touched on last month (generating the XML and creating the XSL file). This time while generating the XML we will add in some of your favorite HTML tags to include images, tables, lists, and some miscellaneous text decoration commands. I will also briefly revisit invoking FOP and displaying the PDF to include a more efficient way of handling these tasks based on reader feedback from last month. I bet you're wondering how FOP knows how to automatically render these HTML tags and why I didn't tell you this before.

This is accomplished by doing a lot of front work in the XSL file using XSLT and XSL-FO. Working with XSL-FO can be complex; in fact, it is one of the largest spec documents at the World Wide Web Consortium, totaling over 400 pages. Don't worry though, you don't have to master XSLT or XSL-FO in order to use this process, nor do you even need to be good with them. One of the beauties of this process is that the XSL is completely separate from your CF application, allowing you to use other resources to create the XSL file. I will try to make this as simple as possible for you by not only demonstrating how to create your own XSL, but also by providing a complete XSL file that will transform the HTML tags that I will discuss in this article.

Sound interesting? Let's get started!

The Example

In this article I will use the results of a simple query on one of the sample databases that come with the ColdFusion install. The following code snippet contains the query I will be using:

```
<cfquery name="getemployees" datasource="exampleapps">
select top 10 firstname,lastname,phone,startdate,title
 from tblEmployees
 where title in ('engineer','web guru')
 order by title,lastname,firstname
</cfquery>
```

The result data will be used to demonstrate tables, unordered lists, and ordered lists. I will use a simple paragraph to demonstrate the other HTML tags.

Preparing for FOP

To create a PDF using FOP requires the input of well-formed XML transformed with XSL-FO. We will generate the XML with ColdFusion by using simple XHTML code wrapped in the <CFXML> tag. To transform the XML we will use the XMLTransform function in ColdFusion and transform it with the contents of a prepared XSL file. The XSL file contains all of the necessary components to format the tags used in the generated XML. Next, we will create the XSL file by integrating XSL-FO and XSLT.

Integrating XSLT and XSL-FO

Let's go back to the basic format of the XSL-FO file that we worked with in part 1. I explained the pieces of the XSL file that FOP expects; be sure to continue to use them. If you are not familiar with this, review part 1 or refer to the complete source code for this article at www.sys-con.com/coldfusion/sourcec.cfm. Last month we used the for-each XSL element to loop over a simple piece of XML code and then used fo:blocks in our loops to format the group of code. That worked well but was very simple and limited. Let's go a little deeper into this part of the process and get a glimpse of what you can become truly of capable of by using FOP to create your PDFs.

Integrating XSLT and XSL-FO allows us to dynamically create all of the XSL-FO needed to create our PDF. We did some of this last time, but in a very limited fashion. Now we will maximize the use of templates to help us do this. Templates are the individual statements that we will use to create the formatting transformation for the simple HTML tags so that FOP can render them.

The following code snippet contains the template I use for the HTML bold tag <strong>. As you can see, the template is wrapped in <xsl:template>. The match attribute of <xsl: template> is used for selecting nodes in an XML string through the use of patterns. A pattern is the syntax we will use to navigate around the XML source tree. Pattern syntax is similar to the command-line syntax used in most operating systems to navigate through directories. As you can see, we are looking for any node matches to the word strong, which will allow us to format anything contained in that node. The code inside the template wrapper can change depending on what you want to do. For bold we use an inline fo style. This could also be any <fo:block> or other XSL-FO element, such as <fo:table>. Then we add the <xsl:apply-templates/> element, which triggers the processing of the children of the node in our XML source that matches the template. These children don't include only other nodes. Elements, attributes, text, and comments are also processed by using the apply-templates element.

```
<xsl:template match="strong">
   <fo:inline font-weight="bold"><xsl:apply-
   templates/></fo:inline>
</xsl:template>
```

Now let's move on to the formatting of the individual HTML tags.

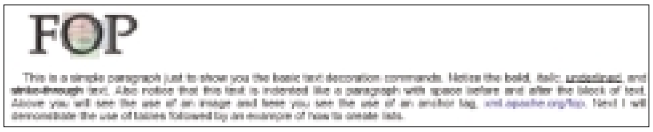
Images, Paragraphs, and Miscellaneous Style Tags

The first tag we will tackle is the image tag; it is much like your everyday <img> tag. You give it a source location of the image and you can specify a height and a width for the image. The current version of FOP requires either an absolute file path or a full URL path. Here, I simply use the ColdFusion function expandPath() to return the absolute file path to the image. The XSL template for img that I have provided looks a little complex, but it's actually a somewhat common code snippet I have seen used to render images. It contains the code needed to handle the image attributes correctly. Here is an example of what a finished XSL-FO image tag might look like.

```
<fo:external-graphic src="#expandPath(logo.jpg)#" height="50px"
width="150px"/>
```

Next is the header (H1), which I will show in detail because I have used an attributes group with it instead of naming the attributes right in the <fo:block>. This method can be useful if there is a common set of attributes that you apply in a few different places, or it can be nice to just define the attributes at the top of the file so that you don't have to go searching for them. Following is the attributes set I have used for the <h1> tag.

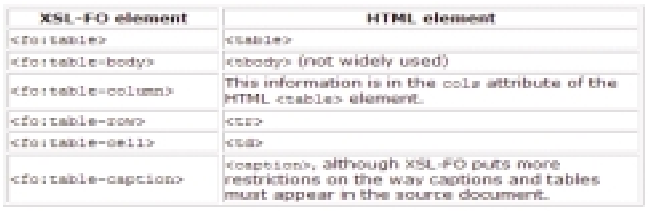
```
<xsl:attribute-set name="h1">
   <xsl:attribute name="font-size">24pt</xsl:attribute>
   <xsl:attribute name="font-weight">bold</xsl:attribute>
   <xsl:attribute name="space-after">10pt</xsl:attribute>
</xsl:attribute-set>
```



FOP

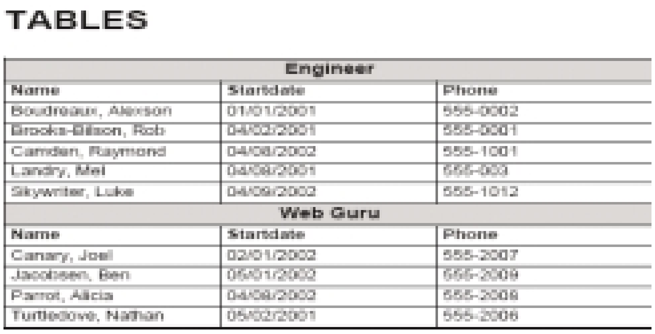
This is a simple paragraph just to show you the basic text decoration commands. Notice the bold, italic, underlined, and ~~strike-through~~ text. Also notice that this text is indented like a paragraph with space before and after the block of text. Above you will see the use of an image and here you see the use of an anchor tag, xml.apache.org/fop. Next I will demonstrate the use of tables followed by an example of how to create lists.

Figure 1: The results of using image, paragraph, and miscellaneous style tags



XSL-FO element	HTML element
<fo:table>	<table>
<fo:table-body>	<tbody> (not widely used)
<fo:table-column>	This information is in the cols attribute of the HTML <table> element.
<fo:table-row>	<tr>
<fo:table-cell>	<td>
<fo:table-caption>	<caption>, although XSL-FO puts more restrictions on the way captions and tables must appear in the source document.

Figure 2: XSLTable elements compared to HTML



TABLES

Engineer		
Name	Startdate	Phone
Boudreaux, Alexson	01/01/2001	555-0002
Brooks-Bilson, Rob	04/02/2001	555-0001
Camden, Raymond	04/08/2002	555-1001
Landry, Mel	04/09/2001	555-003
Skywriter, Luke	04/09/2002	555-1012
Web Guru		
Name	Startdate	Phone
Canary, Joel	02/01/2002	555-2007
Jacobsen, Ben	05/01/2002	555-2009
Parrot, Alicia	04/08/2002	555-2008
Turtledove, Nathan	05/02/2001	555-2006

Figure 3: The result data shown in a table

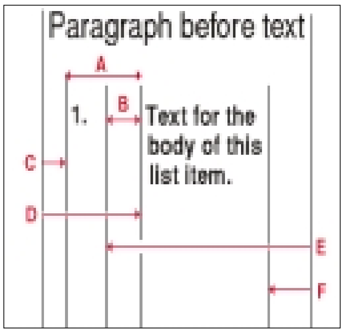
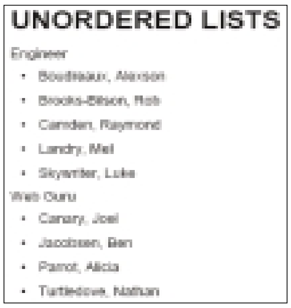


Figure 4: Basic list layout



UNORDERED LISTS

Engineer
- Boudreaux, Alexson
- Brooks-Bilson, Rob
- Camden, Raymond
- Landry, Mel
- Skywriter, Luke

Web Guru
- Canary, Joel
- Jacobsen, Ben
- Parrot, Alicia
- Turtledove, Nathan

Figure 5: The result data shown in an unordered list

With various attribute sets such as this in place, they can be accessed quickly and easily, as shown in the following snippet.

```
<xsl:template match="h1">
   <fo:block xsl:use-attribute-sets="h1"><xsl:apply-
templates/></fo:block>
</xsl:template>
```

Another very useful tag – but one that's a little different from the first two I've shown – is the paragraph tag. In the paragraph template I have added the use of the align attribute. You will notice the xsl:if statement; this allows you to check for an optional attribute and, if it exists, to apply the value by using @nameOfAttribute.

```
<xsl:template match="p">
   <fo:block text-indent="1em" space-before="2pt" space-after="12pt">
      <!-- align attribute -->
      <xsl:if test="@align">
         <xsl:attribute name="text-align"><xsl:value-of
select="@align"/></xsl:attribute>
      </xsl:if>
      <xsl:apply-templates/>
   </fo:block>
</xsl:template>
```

The next tag is the HTML anchor tag. When you see it in the provided code it looks complex; this is because it handles multiple attributes, much like the image tag. It is also a commonly used XSL template for the anchor tag. The difference here is that the anchor tag is used for more than one purpose; the provided code allows it to be used as a local anchor, local destination, and external destination link. The other tags included in the provided code are underline, italic, strikethrough, and break. See Figure 1 for the display results of this first section. For the supporting XSL-FO code for images and anchor tags, refer to the section commented as "HTML Image and Anchor" in Listing 1. If you would like to see all of the XSL-FO code, as well as the basic XHTML code used to make the XML string, please refer to www.sys-con.com/coldfusion/sourcec.cfm.

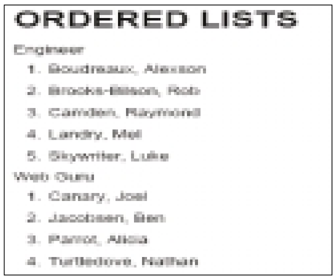
Tables

Tables, as you can imagine, can be complicated. There are a lot of pieces to them, but luckily the tags are very similar to the HTML tags that you are probably used to. The easiest way to see the basic pieces is to look at Figure 2, where they are shown in comparison to their corresponding HTML tags. There are more pieces to the table formatting object, but these are the most commonly used ones.

The code to make the basic table is very similar to the HTML code you may have used to do something similar:

```
<table>
   <tr><th colspan="3" align="center" valign="center">header</th></tr>
   <tr><td>cell 1</td><td>cell 2</td><td>cell 3</td></tr>
</table>
```

The actual XSL-FO table object has a table-body child, which has table-row children, which in turn have table-cell children. One addition to the XSL-FO code to note is the required use of the table-column element. FOP does not automatically figure out how many columns your table has, so a table-column element must be used for each column in your table. By using the XSL for each element and using patterns I was able to figure this out in the table template. You can see the complete code in the tables section of the XSL file in Listing 1. I have created the provided XSL to support colspan, rowspan, align, and valign on both the <th> and <td> tags. The <table> tag will support a width attribute. The tables used in this article are only basic examples. There are many other things that you can do with tables. For example, you can set up table-headers and table-footers that will repeat on breaking pages. To see all of the supporting XSL-FO code for creating tables, refer to the section in Listing 1 commented as “HTML Tables Begin Here”. See Figure 3 for the results of the provided code. To find out more about tables you can refer to chapter 18 of the *XML Bible, Second Edition*, viewable at www.ibiblio.org/xml/books/bible2/chapters/ch18.html.


ORDERED LISTS

Engineer
1. Boudreaux, Alexson
2. Brooks-Bilson, Rob
3. Camden, Raymond
4. Landry, Mel
5. Skywriter, Luke

Web Guru
1. Canary, Joel
2. Jacobsen, Ben
3. Parrot, Alicia
4. Turtledove, Nathan

Figure 6: The data result shown in an ordered list

Lists

Now that you are more familiar with XSL formatting objects, it’s a good time to dip into the details of formatting lists. I will cover unordered and ordered lists in this example. There are four XSL-FO elements used to create a list. An <fo:list-block> has <fo:list-item> children, which have <fo:list-item-label> and <fo:list-item-body> children. Lists can become very complex because of all the different spacing commands available. See Figure 4 to help you understand the different spacing commands and what they do. The following list corresponds to the letters marked in Figure 4:

A. Provisional-distance-between-starts
B. Provisional-label-separation
C. Start-indent for list-item-label
D. Start-indent for list-item-body
E. End-indent for list-item-label
F. End-indent for list-item-body

Unordered lists and ordered lists are very similar. For unordered lists you specify the specific bullet character, which is commonly done using the Unicode character entity, as you can see in the following code snippet.

```
<fo:list-item-label end-indent="label-end()" start-indent="1em">
   <fo:block>&#x2022;</fo:block>
</fo:list-item-label>
```

Ordered lists are done the same way, except that instead of using the bullet character in the <fo:block> to generate the item numbers, you use the <xsl:number> element. To generate numbers you could use <xsl:number format="1. "/>. If you want to have more flexibility with your list items you can use the <xsl:choose> element (just like cfswitch), which allows you to specify a type on your HTML <li> tag. The provided XSL file is set up so that the HTML list commands you already know can be used in the XML to create lists. For the ordered lists I added the logic I mentioned earlier to allow the use of a type attribute on the <li> tag. With the provided XSL file the following type values are available: i, I, a, A, 1. See Figures 5 and 6 to view the output of the provided code. To see the actual XSL-FO code for these results, see the section in Listing 1 commented as “HTML Lists Begin Here”. If you want to get into more detail with lists, I again recommend chapter 18 of the *XML Bible* at www.ibiblio.org/xml/books/bible2/chapters/ch18.html.

Invoking and Displaying the PDF

Now all we need to do is invoke the provided component, fop.cfc. Fop.cfc will call the necessary Java objects included with FOP to create the PDF. Last time we used <cfcontent> to display the PDF file in the browser. After last month’s article was published I received a suggestion on another way to do this that I think is very cool and useful. Neil Giarratana sent me an e-mail with instructions on how to stream the PDF into the browser without ever creating a PDF file. However, there are pros and cons to this method. The main drawback is that it is not an officially supported technique. If you use this method, make sure to encapsulate it well so that it will be easy to change if the method is not functional in future versions of ColdFusion. The major pro of this method is that there is no need to access the directory on the server to save the PDF file. I have included the necessary code for this method in this article, but if you do not want to use it and would prefer to use a supported method, refer to the source code for part 1 at www.sys-con.com/story/?storyid=44771&DE=1.

Neil's alternative method involves changing the called method of the java.io package in fop.cfc. In the previous version of fop.cfc we used the FileOutputStream method, but now we will use the ByteArrayOutputStream method. I have included the modified version of fop.cfc in Listing 2. When invoking fop.cfc there is no longer a need to use the PDFFile argument since it is not actually creating the PDF file. Instead, a ReturnVariable argument is added. In place of the <cfcontent> tag we used last time, we will now use a little block of cfscript that Neil sent me (see Listing 3). There is nothing wrong with the way we did this last time; as I mentioned, both methods have their pros and cons. This is just another great way to create a PDF.

Note: The complete source code for this article is too long to print, but is available at www.sys-con.com/coldfusion/sourcec.cfm.

Conclusion

Now you not only know how to create a free PDF, you can also make your results very presentable simply by using the HTML tags that you already know. Hopefully, you can now see the possibilities of this process and how useful it can be. There is still so much more that can be done with this process. If you read up on some resources for XSL-FO, you will see what I mean. There is no way I could completely cover everything in just a few articles. I hope to encourage more people to get started using this process. Hopefully, others will chime in with how they have used it. If you have any suggestions or comments on it, send them my way at natenelson@comcast.net. In future articles in this series I may eventually touch on encryption, tables of contents, indexes, coversheets, or SVG support. Feel free to send suggestions or comments on what you would most like to see. I look forward to hearing from you, but most important, have fun!

Resources

- *The XML Bible: Second Edition:* www.ibiblio.org/xml/books/bible2/chapters/ch18.html
- *The FOP project:* http://xml.apache.org/fop
- Pawson, D. (2002). *XSL-FO Making XML Look Good in Print.* O'Reilly.
- *XSL FO reference:* http://zvon.org/xxl/xslfoReference/Output/index.html

About the Author

*Nate Nelson is currently a senior software developer for I*LEVEL, Inc., a software startup out of Denver, Colorado, where he leads development efforts of a commercial software product. He has worked with several nonprofit organizations and commercial corporations to architect applications to address many diverse needs. Nate is currently a member of the Denver CFUG administration.*

natenelson@comcast.net

Listing 1

```
  <!-- ========================
  HTML Image and Anchor
  ============================ -->

 <!-- HTML anchor tag, allows local anchor and external href -->
 <xsl:template match="a">
   <xsl:choose>
      <!-- If this is a named anchor, create an empty block with id -->
     <xsl:when test="@name">
       <xsl:if test="not(name(following-sibling::*[1]) = 'h1')">
         <fo:block line-height="0pt" space-after="0pt"
           font-size="0pt" id="{@name}"/>
       </xsl:if>
     </xsl:when>
     <xsl:when test="@href">
       <fo:basic-link color="blue">
         <xsl:choose>
            <!-- if href starts with # then local anchor -->
           <xsl:when test="starts-with(@href, '#')">
             <xsl:attribute name="internal-destination">
               <xsl:value-of select="substring(@href, 2)"/>
             </xsl:attribute>
           </xsl:when>
           <!-- else, external hyperlink -->
           <xsl:otherwise>
             <xsl:attribute name="external-destination">
               <xsl:value-of select="@href"/>
             </xsl:attribute>
           </xsl:otherwise>
         </xsl:choose>
         <xsl:apply-templates select="*|text()"/>
       </fo:basic-link>
       <xsl:if test="starts-with(@href, '#')">
         <xsl:text> on page </xsl:text>
         <fo:page-number-citation ref-id="{substring(@href, 2)}"/>
       </xsl:if>
     </xsl:when>
   </xsl:choose>
 </xsl:template>

  <!-- Basic HTML image tag, allows height and width -->
 <xsl:template match="img">
    <fo:block space-after="12pt">
      <fo:external-graphic src="{@src}">
        <xsl:if test="@width">
          <xsl:attribute name="width">
            <xsl:choose>
              <xsl:when test="contains(@width, 'px')">
                <xsl:value-of select="@width"/>
              </xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="concat(@width, 'px')"/>
              </xsl:otherwise>
          </xsl:choose>
        </xsl:attribute>
      </xsl:if>
      <xsl:if test="@height">
        <xsl:attribute name="height">
          <xsl:choose>
            <xsl:when test="contains(@height, 'px')">
              <xsl:value-of select="@height"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="concat(@height, 'px')"/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:attribute>
      </xsl:if>
    </fo:external-graphic>
  </fo:block>
</xsl:template>

 <!-- ======================
 HTML Tables begin here
 ========================== -->
 <!-- table tag -->
 <xsl:template match="table">
    <fo:table xsl:use-attribute-sets="table.data">
       <xsl:if test="@layout">
          <xsl:attribute name="table-layout">
             <xsl:value-of select="@layout"/>
          </xsl:attribute>
       </xsl:if>
       <xsl:if test="@width">
          <xsl:attribute name="inline-progression-dimension">
             <xsl:value-of select="@width"/>
          </xsl:attribute>
       </xsl:if>
       <xsl:for-each select="tr[1]/th|tr[1]/td">
          <fo:table-column>
             <xsl:if test="@colspan">
                <xsl:attribute name="number-columns-repeated">
                   <xsl:value-of select="@colspan"/>
                </xsl:attribute>
             </xsl:if>
             <xsl:attribute name="column-width">
                <xsl:value-of select="floor(@width div 72)"/>in
             </xsl:attribute>
          </fo:table-column>
       </xsl:for-each>
       <fo:table-body>
          <xsl:apply-templates/>
       </fo:table-body>
   </fo:table>
 </xsl:template>

 <!-- Table Row -->
 <xsl:template match="tr">
    <fo:table-row>
       <xsl:apply-templates/>
    </fo:table-row>
 </xsl:template>

 <!-- HTML table tag <th> -->
 <xsl:template match="th">
    <fo:table-cell xsl:use-attribute-sets="table.data.th">
       <!-- call cell-span template to allow colspan and rowspan
attributes -->
       <xsl:call-template name="cell-span"/>
       <!-- valign attribute -->
         <xsl:if test="@valign">
            <xsl:attribute name="display-align">
               <xsl:value-of select="@valign"/>
            </xsl:attribute>
         </xsl:if>
         <fo:block>
            <!-- align attribute -->
            <xsl:if test="@align">
               <xsl:attribute name="text-align">
                  <xsl:value-of select="@align"/>
               </xsl:attribute>
            </xsl:if>
            <xsl:apply-templates/>
         </fo:block>
      </fo:table-cell>
   </xsl:template>

   <!-- HTML table tag <td> -->
   <xsl:template match="td">
      <fo:table-cell xsl:use-attribute-sets="table.data.td">
         <xsl:call-template name="cell-span"/>
         <xsl:if test="@valign">
            <xsl:attribute name="display-align">
               <xsl:value-of select="@valign"/>
            </xsl:attribute>
         </xsl:if>
         <fo:block>
            <xsl:if test="@align">
               <xsl:attribute name="text-align">
                  <xsl:value-of select="@align"/>
               </xsl:attribute>
            </xsl:if>
            <xsl:apply-templates/>
         </fo:block>
      </fo:table-cell>
   </xsl:template>

   <!-- This is called in other templates to allow rowspan and colspan
attributes -->
   <xsl:template name="cell-span">
     <xsl:if test="@colspan">
       <xsl:attribute name="number-columns-spanned">
         <xsl:value-of select="@colspan"/>
       </xsl:attribute>
     </xsl:if>
     <xsl:if test="@rowspan">
       <xsl:attribute name="number-rows-spanned">
         <xsl:value-of select="@rowspan"/>
     </xsl:attribute>
     </xsl:if>
   </xsl:template>

   <!-- ======================
   HTML Lists Begin here
   ======================== -->
   <!-- Unordered Lists -->
   <xsl:template match="ul">
          <fo:list-block space-before="0.25em" space-after="0.25em">
              <xsl:apply-templates/>
          </fo:list-block>
   </xsl:template>

   <xsl:template match="ul/li">
       <fo:list-item xsl:use-attribute-sets="list.item">
         <fo:list-item-label end-indent="label-end()" start-
indent=".75em">
           <fo:block>&#x2022;</fo:block>
          </fo:list-item-label>
          <fo:list-item-body start-indent="body-start()">
            <fo:block>
              <xsl:apply-templates/>
            </fo:block>
          </fo:list-item-body>
        </fo:list-item>
  </xsl:template>

   <!-- Ordered Lists -->
   <xsl:template match="ol">
        <fo:list-block space-before="0.25em" space-after="0.25em">
            <xsl:apply-templates/>
        </fo:list-block>
   </xsl:template>

  <xsl:template match="ol/li">
        <fo:list-item xsl:use-attribute-sets="list.item">
          <fo:list-item-label end-indent="label-end()" start-
indent=".75em">
            <fo:block>
               <!-- This allows type attribute to specify the number
format -->
              <xsl:choose>
                <xsl:when test="@type='i'">
                  <xsl:number format="i. "/>
                </xsl:when>
                <xsl:when test="@type='I'">
                  <xsl:number format="I. "/>
                </xsl:when>
                <xsl:when test="@type='a'">
                  <xsl:number format="a. "/>
                </xsl:when>
                <xsl:when test="@type='A'">
                  <xsl:number format="A. "/>
                </xsl:when>
                <xsl:otherwise>
                  <xsl:number format="1. "/>
                </xsl:otherwise>
              </xsl:choose>
            </fo:block>
          </fo:list-item-label>
          <fo:list-item-body start-indent="body-start()">
            <fo:block>
              <xsl:apply-templates/>
            </fo:block>
          </fo:list-item-body>
        </fo:list-item>
  </xsl:template>
```

Listing 2

```
FOP.cfc
<cfcomponent>
   <!--- Initialize the FOP driver from xml.apache.org --->
   <cfset variables.driver = CreateObject("java",
"org.apache.fop.apps.Driver")>
   <cfset variables.lockname = CreateUUID()>

   <cffunction name="ConvertFileToPDF" access="public">
      <cfargument name="foFile" type="string" required="true">
      <cfset var output = "" />
      <!--- Set up Java input source --->
      <cfset var input = CreateObject("java",
"org.xml.sax.InputSource")>
      <cfset input.init( ARGUMENTS.foFile)>
      <!--- Proceed with PDF generation --->
      <cfset output = private_generatePDF(input)>
      <cfreturn output />
   </cffunction>

   <cffunction name="ConvertStringToPDF" access="public">
      <cfargument name="foString" type="string" required="true">
      <!--- Set up Java input source --->
      <cfset var reader = CreateObject("java",
"java.io.StringReader")>
      <cfset var input = CreateObject("java",
"org.xml.sax.InputSource")>
      <cfset reader.init(foString)>
      <cfset input.init(reader)>
      <!--- Proceed with PDF generation --->
      <cfset output = private_generatePDF(input)>
      <cfreturn output />
   </cffunction>

   <cffunction name="private_generatePDF" access="private">
      <cfargument name="input" type="any" required="true">
      <!--- Output stream for writing PDF file --->
      <cfset var output = CreateObject("java",
"java.io.ByteArrayOutputStream")>
      <!--- Turn on the output stream --->
      <cfset output.init()>
      <!--- Hook the FOP driver to the input/output --->
      <cfset variables.driver.setInputSource(input)>
      <cfset variables.driver.setOutputStream(output)>
      <!--- Perform the actual PDF generation --->
      <cfset variables.driver.run()>
      <cfreturn output />
   </cffunction>
</cfcomponent>
```

Listing 3

```
modified piece of TransdformXML.cfm
<!--- now invoke the FOP component - pass in transformed xml and pdf
file to be created --->
<cfinvoke component="FOP" method="ConvertStringToPDF"
foString="#TransformedXMLCode#" returnvariable="FOPByteArray">

<!--- Now display PDF file to browser --->
<cfscript>
   //Retrieve the page context's response object
   context = getPageContext();
   context.setFlushOutput(false);
   response = context.getResponse().getResponse();
   out = response.getOutputStream();
   //Set the MIME type
   response.setContentType("application/pdf");
   //We must specify the content length but that's easy using the
ByteArrayOutputStream's size method
   response.setContentLength(FOPByteArray.size());
   //Write the output to the browser and flush (to actually send)
   out.write(FOPByteArray.toByteArray());
   out.flush();
   //Close the output stream object
   out.close();
</cfscript>
```

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

One damn good Cold Fusion hosting firm!

Webcore Technologies specializes in ColdFusion, ASP and SQL hosting. We offer these solutions in both dedicated server and shared virtual environments. Webcore can design and implement clustered or load-balanced solutions, managed firewalls, VPN's, multi-site failover, and more. In addition, we also offer corporate Internet access, web site design, and technology consulting services.

Our in-house tier 1 level data center enables us to provide the most reliable hosting in the industry. Our Microsoft and Cisco certified team ensures that all support issues are resolved promptly and professionally. Unlike other hosting companies that answer with a phone attendant, you will receive a live person when you call Webcore. No phone attendant, no frustrations, just world class customer service and support the first time you call.

Cisco Systems

Microsoft Certified Partner

Webcore Technologies

Webcore Technologies, Inc.
877.WCT.HOST
www.webcoretech.com

be the pilot!

FREE SETUP on Shared Hosting Accounts With ColdFusion MX Support
Use Promo Code CFDJ2004

INTERMEDIA.NET

WE DARE YOU TO TAKE A FREE TEST FLIGHT!

Managing technology that runs your business is a matter of trust and control. INTERMEDIA.NET gives you both.

TRUST. Since 1995 we have been providing outstanding hosting service and technology to our clients. Don't take our word for it... take theirs.

"The support and service that you offer are nothing short of golden. The high quality of your system and service for CF customers is something one could only ever dream of." – Claude Raiola, Director, AustralianAccommodation.com Pty. Ltd.

CONTROL. We give you instant control over your site, server and account configuration changes. No more submitting requests and waiting for someone else to take action. You are in control to pilot your business through its daily needs.

BE THE PILOT. Take a free test flight and see what our HostPilot™ Control Panel offers you beyond all others. Check out our SLA guarantees. To see more testimonials and to find out about our competitive advantages, visit our Web site at www.Intermedia.NET.

Managed Hosting • Shared Hosting • Microsoft Exchange Hosting

Call us at: 1.800.379.7729 • Visit us at: WWW.INTERMEDIA.NET

HostPilot™