

ColdFusion Developer's Journal

ColdFusionJournal.com

September 2002 Volume: 4 Issue: 9

web servicesEDGE
world tour 2002

COMING TO A CITY NEAR YOU

SEE PAGE 39 FOR DETAILS

2002

LOS ANGELES ---- SEPTEMBER 19

SAN JOSE ----- OCTOBER 3

CHICAGO ----- OCTOBER 17

ATLANTA ----- OCTOBER 29

MINNEAPOLIS --- NOVEMBER 7

AND MANY MORE!

Editorial

Tying It All Together...

Robert Diamond page 5

Q&A

Ask the Training Staff

page 46

CF Community

Tales from the List

Simon Horwith page 48

CFDJ News

Macromedia's Pet Market Application

page 50

SYS-CON
MEDIA



by Hal Helms
page 6

<BF>on<CF>: Undocumented ColdFusion MX 1.0	12
<i>Discover the thrill of sailing in uncharted waters</i>	Ben Forta
Foundations: Managing Your Career in Tough Times	18
<i>The wisdom of planning for the future</i>	Hal Helms
CFMX Migration: An Insider's View	20
<i>Tips for a trouble-free migration to ColdFusion MX</i>	Jim Schley
Journeyman ColdFusion: Hidden Gems in CFMX	24
<i>Mining the available resources</i>	Charles Arehart
Feature: Creating Maintainable Web Sites	28
<i>Implementing MVC patterns with CF components</i>	David Gassner
Product Review: ColdFusion by Macromedia	34
<i>The Web's new fax machine?</i>	Carey Lilly
Tracking: Tracking Traffic at Your Site	38
<i>Using a database and CF's graphing capability</i>	David Shadovitz
Best Practices: CFC Best Practices & Tips	42
<i>Guidelines for working with CF components</i>	Raymond Camden

INTERLAND
www.interland.com

NEW ATLANTA
www.newatlanta.com

ACTIVEPDF
www.activepdf.com

editorial advisory board

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

editorial

editor-in-chief

Robert Diamond robert@sys-con.com

editorial director

Jeremy Geelan jeremy@sys-con.com

executive editor

M'lou Pinkham mpinkham@sys-con.com

managing editor

Cheryl Van Sise cheryl@sys-con.com

editor

Nancy Valentine nancy@sys-con.com

associate editors

Jamie Matusow jamie@sys-con.com

Gail Schultz gail@sys-con.com

Jean Cassidy jean@sys-con.com

assistant editor

Jennifer Stilley jennifer@sys-con.com

production

vp, production & design

Jim Morgan jim@sys-con.com

lead designer

Cathryn Burak cathyb@sys-con.com

art director

Alex Botero alex@sys-con.com

associate art directors

Louis F. Cuffari louis@sys-con.com

Richard Silverberg richards@sys-con.com

Aarathi Venkataraman aarathi@sys-con.com

assistant art director

Tami Beatty tami@sys-con.com

contributors to this issue

Charles Arehart, Raymond Camden,
Ben Forta, David Gassner, Hal Helms,
Simon Horwith, Carey Lilly, Jim Schley,
David Shadovich, Bruce Van Horn

editorial offices

SYS-CON MEDIA

135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645

TELEPHONE: 201 802-3000 FAX: 201 782-9600

COLDFUSION DEVELOPER'S JOURNAL (ISSN #1523-9101)

is published monthly (12 times a year)

by SYS-CON Publications, Inc.

postmaster: send address changes to:

COLDFUSION DEVELOPER'S JOURNAL

SYS-CON MEDIA

135 Chestnut Ridge Rd., Montvale, NJ 07645

©copyright

Copyright © 2002 by SYS-CON Publications, Inc.
All rights reserved. No part of this publication may be
reproduced or transmitted in any form or by any means,
electronic or mechanical, including photocopy or any
information, storage and retrieval system,
without written permission.

For promotional reprints, contact reprint coordinator.
SYS-CON Publications, Inc., reserves the right to revise,
republish and authorize its readers to use the articles
submitted for publication.

All brand and product names used on these pages
are trade names, service marks, or trademarks
of their respective companies.



Tying It All Together...Macromedia Style



BY ROBERT DIAMOND

One of the ever-repeating themes here at **CFDJ** since Macromedia and Allaire merged has been the converging of their technologies and our place in the ever-changing technology landscape as CF developers. We spoke about it first while looking into the future with a sense of hope when the merger took place. Then we covered it as details began to emerge on what they had planned.

Since then, as **CFDJ** evolves along with the ColdFusion market, we've continued to cover all sorts of cross uses of these technologies. CF and Flash, CF and Java, and other such combo possibilities are increasingly becoming part of our collective development projects. The choices on what to use on most programming projects are becoming as varied and as long as a Chinese takeout menu. Further complicating things, Microsoft is working very hard to make sure that the next time I write about convergence my first example is CF and .NET.

I got some interesting feedback to my editorial last month, in which I attempted to roughly map out ColdFusion's place in the *i*-technology universe. Most readers agreed, more or less, with my choice in positioning, and e-mails I received on the subject sparked a lot of lively discussions. The majority also felt I was right in saying that Macromedia was doing their part to keep ColdFusion in that spot, their commitment to this being critical to our desire and confidence about investing in their products.

Macromedia's mission, under the MX umbrella, is to help developers toward their goals of creating the best possible applications, with the least amount of nightmares along the way, and also, of course, staying competitive as workers in the developer marketplace.

Simply being a ColdFusion developer is a good start on the mission, but it's just a start. I've spoken to lots of folks recently who have been unsure of where to start to expand their base of knowledge. Luckily for them, and even for advanced folks who are also branching out, Macromedia has released a good new resource, their Pet Market application.

There are actually four separate versions of the Pet Market application: two Flash front ends, one for Microsoft's .NET Pet Shop and one for Sun's Java Pet Store. More interesting to CFers than the Java versus .NET debate that rages around us are the two blueprint applications: one for Windows, one for Linux/Unix (another dangerous topic of debate). The latter are built using a combination of CF and Flash, and are worth some time playing with if you're ready to jump into these new waters. They use ColdFusion components (CFCs), which are my favorite new feature in ColdFusion MX, so I'm especially partial to seeing good examples used to get the word out. It's also got some good uses of a lot of the time-saving features of the new Dreamweaver, which I'm happy to say has simplified my development life.

It's available in Macromedia's Designer/Developer center, so go play with that – and read on – we've got a great issue!



Robert Diamond

@ ROBERT@SYS-CON.COM

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



ColdFusion Feature

HAL HELMS

A Book Excerpt

This article is based on the upcoming *Discovering ColdFusion Components* by Hal Helms, published by Techspedition Press (www.techspedition.com), and appears here by permission of the publisher.

Specialization: A powerful means of avoiding duplication of code

ColdFusion components (CFCs) have generated more discussion than any other aspect of the newly released ColdFusion MX server. At one end of the spectrum of opinion is the disparaging view of some that CFCs are “nothing more” than structures with user-defined functions (UDFs) attached. Equally extreme is the idea held by an impassioned few that with CFCs, object-oriented programming has come to ColdFusion and will sweep away everything that has preceded it.

ColdFusion components have many similarities to objects in OO languages, but they are still quite different. For now let's simply take a look at them on their own terms, for however we may decide to classify them, CFCs are extraordinarily useful.

CFCs are particularly valuable in letting developers create a software model for a category of things that exist in the world. That category can either be tangible – persons, cars, buildings – or conceptual – companies, tasks, agreements. Because concrete examples are often easier to learn, let's start our exploration by creating a CFC that models a person.

Start by creating a file named Person.cfc and save it under your Web root. (All examples given can be downloaded from www.sys-con.com/coldfusion/sourcec.cfm or www.techspedition.com. You'll find it easiest if you create a directory structure of com/techspedition/CFCs directly under your Web root. Place Person.cfc into the CFCs folder. If you use another folder structure, you'll need to change the code presented here to accommodate a different directory layout.)

In your Person.cfc file add the <cfcomponent> tag in its barest of forms:

```
<cfcomponent>
</cfcomponent>
```

That code provides a container for all subsequent code related to our Person CFC. Notice that it has no name attribute to the tag. It knows who it is because only one CFC is permitted per file. Therefore, the name of the file determines the name of the CFC. If you find it disconcerting not to see a name in the <cfcomponent> tag (I do), you can use the hint attribute of <cfcomponent> as in

```
<cfcomponent hint="I am a Person">.
(That attribute provides the tag with information about itself known as metadata.)
```

CFCs are like objects in that a CFC is primarily a “thing” (rather than an action or process). We are used to things having properties: a car has a make, a model, a price tag, a color, and so forth. Each CFC has a built-in structure called “this” that can be used for storing the values of properties.

Let's add the following properties to this, setting their value to null:

- gender="null"
- firstName="null"
- lastName="null"
- dateOfBirth="null"

Note that in ColdFusion MX, unlike some languages, *null* is not a special value or reserved word – it's simply a string. We could have used any string we chose to, including an empty string.

Your file, Person.cfc, should look like this:

```
<cfcomponent>
<cfset this.gender = "null">
<cfset this.firstName = "null">
<cfset this.lastName = "null">
<cfset this.dateOfBirth = "null">
</cfcomponent>
```

There are several ways of calling a CFC, including using the <cfinclude> tag and calling the CFC as the action page of a form. For now, let's use the CreateObject() function.

Create a page called Tester.cfm and save this in your CFCs directory. On this page create a new Person named matt using this code:

```
<cfset matt = CreateObject( 'component', 'com.techspedition.CFCs.
    Person' )>
<cfdump var="#matt#">
```

You should see something like the screen shown in Figure 1.

The <cfdump> tag has built into it the ability to output complex variables in an onscreen hierarchy. The variable, matt, is the structure, this. When you created the matt object using CreateObject(), that object was returned to you; matt is simply a handle on the object created.

You can output the value of any of matt's properties. Remove the <cfdump> tag from Tester.cfm and insert this code in its place:

```
<cfoutput>
#matt.firstName#
</cfoutput>
```

What you get back is the default value, null, which was set previously using the <cfparam> tag within Person.cfc.

In the OO worlds of Smalltalk and Java, it's often said that an object's properties represent everything an object knows about itself, while an object's methods characterize everything an object can do. If we judge by this standard, we must admit that matt is a little “slow.” He knows some things about himself but can't do anything. Let's change that by introducing some methods.

The first method will help us recover from some bad advice I gave you. I had you output matt's firstName by directly accessing it as matt.firstName. One of the principles of OO programming is called *encapsulation*. Only the object should know its internal state and should directly access its variables. Everyone else should send a request to the object for that information. This mechanism allows the object to decide what information to provide and how to present it.

In Java we can enforce encapsulation by designating variables as private, meaning that they can't be directly accessed – unlike our code that dereferenced matt's firstName. We can't protect CFCs in the same manner, but we can encourage others to make requests of the object by providing methods that supply the needed information.

Let's create a method for Person.cfc called getFirstName(). All methods are created using the <cffunction> tag within a CFC. To create getFirstName(), place this code directly beneath the <cfset> tags you already wrote for Person.cfc.

```
<cffunction name="getFirstName">
<cfreturn this.firstName>
</cffunction>
```

The function, as you can see, is very simple. It does nothing more than use a <cfreturn> tag to return the value of this.firstName to the requester. To make use of this new function, create Tester2.cfm. Place the code to create matt again and then call matt.getFirstName():

```
<cfset matt = CreateObject( 'component', 'Person' )>
<cfoutput>
#matt.getFirstName()#
</cfoutput>
```

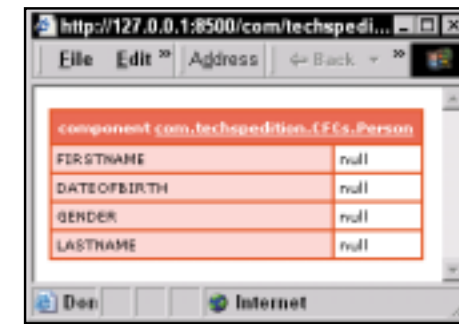


FIGURE 1 Outputting the matt object

If you run this code, you will again get a value of null, just as you did above. This time, though, we're using a method to access information rather than prying into the internals of a CFC.

We've made matt a little more capable, but his knowledge of himself is lacking – everything is null. Let's change that by creating a new() method for Person.cfc that will accept several arguments passed into it. A new method means another <cffunction> tag set. Here's the code for new():

```
<cffunction name="new">
<cfargument name="firstName"
    required="No" type="string"
    default="null">
<cfargument name="lastName"
    required="No" type="string"
    default="null">
<cfargument name="gender"
    required="No" type="string"
    default="null">
<cfargument name="dateOfBirth"
    required="No" type="string"
    default="null">
```

```
<cfset this.firstName =
arguments.firstName>
<cfset this.lastName =
arguments.lastName>
<cfset this.gender = arguments.gender>
<cfset this.dateOfBirth = arguments.
dateOfBirth>
</cffunction>
```

The <cfargument> tags alert individual methods that arguments may be passed into them. Arguments passed into a method are available to that method in a construct called arguments.

A good argument can be made that all code that deals with creating a new object should be encapsulated inside a method, rather than simply sitting unattached inside the <cfcomponent> tags as our four <cfset> tags are now doing. Since I always create and call a new() method in order

to initialize a CFC, I've done just that, removing the <cfset> tags and using instead the two attributes of the <cfargument> tag.

The required="No" snippet is used to indicate that the argument is optional; the default="null" snippet will provide values if Person's new() method is called without passing arguments into it. To make use of this code, write Tester3.cfm to contain this code:

```
<cfset matt = CreateObject( 'component', 'Person' )>
<cfset matt.new( 'Matt', 'Cox', 'Male', 'Feb 30, 1981' )>
<cfdump var="#matt#">
```

You should see something similar to Figure 2.



FIGURE 2 Displaying the matt object with <cfdump>

The arguments passed into the CFC are matched in order with the <cfargument> tags in the CFC method. In OO languages you can have duplicate method definitions so long as the number and/or data types of the arguments differ. Such a mechanism, known as overloading methods, is legal in Java but not allowed in CFCs.

The display in Figure 2 shows both the properties and the methods available to matt. In order to better support encapsulation, I've provided the Person CFC with additional functionality, including methods to get and set individual property information, commonly called getters and setters. The code for the finished

component can be seen in Listing 1, downloadable, as are all the listings, from www.syscon.com/coldfusion/sourcec.cfm (or from www.techspedition.com).

One thing that may not have been obvious is that while Person.cfc was used to create matt, the Person.cfc is different from matt. You might think of Person.cfc as a sort of factory, creating objects that have their own existence. This becomes more obvious if we give matt a mate, using the same Person.cfc that created matt. Create Tester4.cfm with this code:

```
<cfset matt = CreateObject( 'component', 'Person' )>
<cfset matt.new( 'Matt', 'Cox', 'Male', 'Feb 30, 1981' )>
<cfdump var="#matt#">
```

```
<cfset mandy = CreateObject( 'component', 'Person' )>
<cfset mandy.new( 'Mandy', 'Cox', 'Female', 'April 1, 1976' )>
<cfdump var="#mandy#">
```

Run this code and you'll see that matt and mandy are distinct from one another. If you want to make a change to one, the other will be unaffected.

```
<cfset matt.setFirstName( 'Stan' )>
<cfoutput>
```

Matt used to be Matt but changed his name to #matt.getFirstName()#; his mate, #mandy.getFirstName()#, stayed with her man.

```
</cfoutput>
```

Run this code and you'll see some inspired verse: "Matt used to be Matt but changed his name to Stan, but his mate, Mandy, stayed with her man."

Let's leave such deathless poetry aside and look at a quick primer on object relationships.

Objects can stand in different relationships to one another. Some objects really have nothing to do with each other – Person and Ocean, for example (see Figure 3).

On the other hand, some objects are related to each other by means of aggregation or composition. For example, a Person might have an Address (see Figure 4).

This relationship is often called "has-a" – as in "a Person has a Address".

Still, a third relationship is called specialization, in which one object is a specialized form or type of another object. This relationship is often called "is-a" – as in "an Employee is a Person" (see Figure 5).

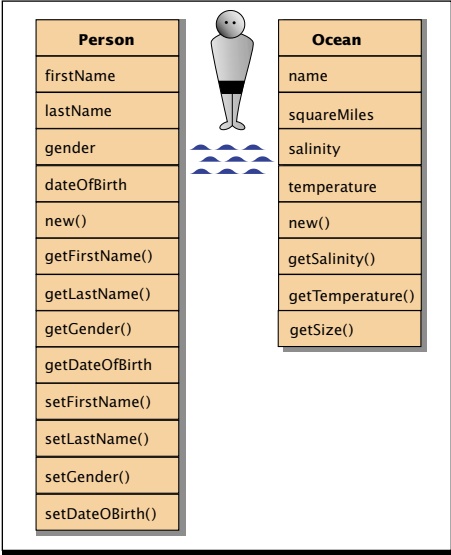


FIGURE 3 Unrelated objects

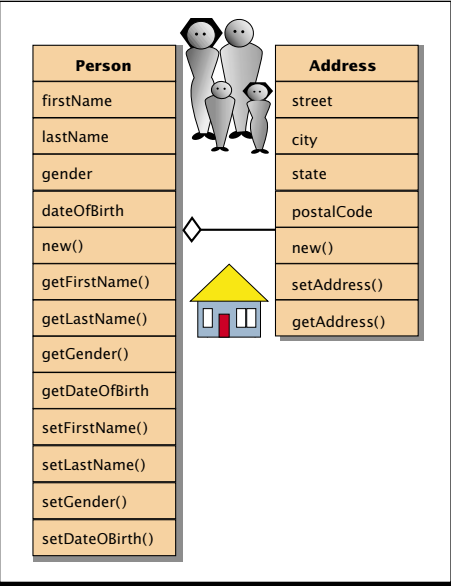


FIGURE 4 A Person "has a" Address

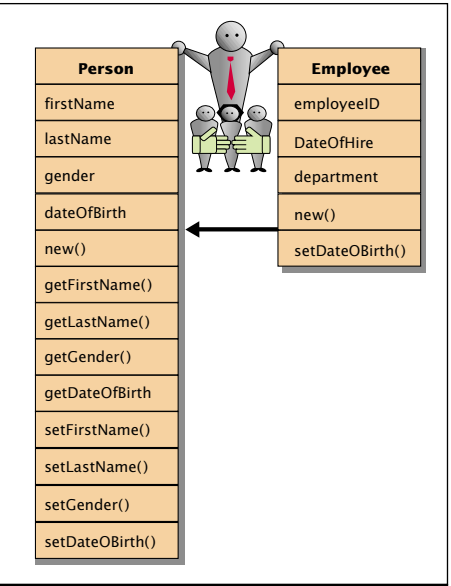


FIGURE 5 An Employee "is a" Person

MACROMEDIA

www.macromedia.com/go/cfmxad

The object that is the more general form (Person, in this example) is known as a supertype; the more specialized form (Employee) is called a subtype. In both OO languages and ColdFusion's CFCs, subtypes are given the special ability to inherit from their supertypes.

Inheritance allows subtypes to "borrow" functionality and properties from their supertypes. For example, if we have a CFC called Sportscar that extends Car, if Car already has defined properties for make, model, and color, Sportscar doesn't need to duplicate these (although it may choose to change the values of some of its parent's variables). The same is true for methods – methods in the supertype are inherited by the subtype. If Car has methods drive() and stop(), then Sportscar will inherit these methods.

To mark a CFC as a subtype of another CFC, you use the extends attribute of the <cfcomponent> tag. The value of extends should be the name of the CFC you wish to inherit from or extend. Any CFC wishing to be a subtype of Person requires this code:

```
<cfcomponent extends="com.techspedition.CFCs.Person">
```

To illustrate this, let's create Employee.cfc, which extends Person.cfc. (The code for this can be found in Listing 2.) Once you've created an employee "factory," let's create a real employee, mandy.

Create Tester5.cfm. Insert (or download) the code shown in Listing 3 and execute it. The last line in this code <cfdump>s the Employee object, mandy, to the screen. The results are shown in Figure 6 (methods have been visually collapsed to save space).

Notice that because mandy is an Employee, she inherited the methods of Employee's superclass, Person. This is inheritance at work. But something is missing. When we created a new Person previously, it had properties such as firstName, lastName, etc. But those aren't there in our new Employee, mandy. Why not?

If you think about this, you'll see that those properties were set when we called Person.new(). But in creating mandy, we called Employee.new(). In other words, we have a new() method in both Employee.cfc and in Person.cfc. When both subtype and supertype have the same named method, the subtype is said to have overridden the supertype for that particular method.

Calling mandy.new() will execute the method code for new() in Employee.cfc,

not in Person.cfc. That's true with both CFCs and OO languages. However, OO languages have a mechanism that nicely skirts the problem: a subtype can call its supertype to execute the supertype's version of the overridden method before the subtype method executes its own.

It can do this because each subtype object can refer to its supertype by means of the variable, super. In OO languages, super is automatically created. This mechanism of a subtype calling its super to do some work is very common. Unfortunately, CFCs don't have the concept of super. This omission has undesirable consequences. If you try to get mandy's birthdate, as in mandy.getDateOfBirth(), what you'll get instead is an error: Element DATEOFBIRTH is undefined in THIS.

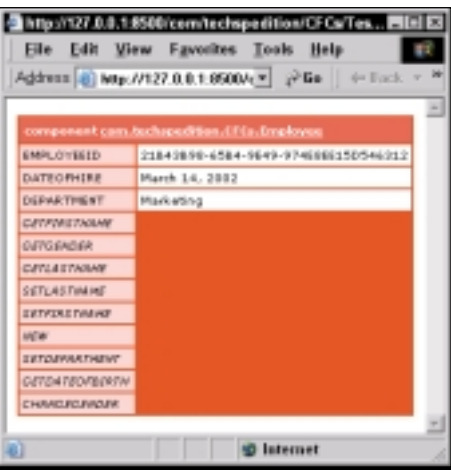


FIGURE 6 Displaying mandy, an Employee that inherits from Person

Getting such an error makes sense – the data element, dateOfBirth, as well as gender, firstName, and lastName, are only created during the invocation of Person.new(), and since Employee overrides the new() method, Person.new() will never be called.

In the absence of a built-in concept of super, we can build our own. Listing 4 defines a new Employee.cfc. Note that the first code to execute is the definition of a new object, super, that points to the supertype, Person.cfc. With the idea of super established, the body of Employee's new() method can make a call to super:

```
<cfinvoke
component="#super#"
method="new"
subtype="#this#"
returnvariable="aPerson"
firstName="#arguments.firstName#"
lastName="#arguments.lastName#"
gender="#arguments.gender#"
dateOfBirth="#arguments.
dateOfBirth#">
```

```
dateOfBirth="#arguments.
dateOfBirth#">
```

I've implemented the call to super as a <cfinvoke> tag, though it could also be coded as super.new(). The advantage the <cfinvoke> tag has is that it is much easier to tell the name/value pairs being passed. When using the Object.method() syntax – such as super.new() – values are passed without accompanying names, making them harder for a human reader to understand.

The <cfinvoke> code sends just those arguments to Person.new that are needed – firstName, lastName, gender, and dateOfBirth. We expect something to be returned from the call to the super's new() method and specify the variable name to be returned using the returnvariable attribute. Notice that we are sending this along with the other variables in a variable called subtype. To see how this works, look at the code for Person.cfc, revised to work with this call. This code is shown in Listing 5.

The new code in Person.cfc is the <cfif> code block:

```
<cfif IsDefined( 'arguments.
subType' )>
<cfset child=arguments.subType>
<cfset StructAppend( child, this,
'no' )>
<cfreturn child>
</cfif>
```

Person.new() can be called in two ways. We saw the first way when we created matt, using a matt.new() syntax. Now we introduce another way – the subtype calling Person.new(). If Person.new is called by a subtype, the following instructions should be executed:

1. Create a local variable, child, and point it to the subType argument passed into the Person object.
2. Append the properties in the Person object's this structure to the child structure.
3. Return the child, which will be the result of taking the Employee object's this structure (passed in as subType) and appending the Person object's this structure. The result is a combination of the properties of Person and Employee.

If, though, Person.new() is being called normally – as we did when we created matt – this code should not execute. The Person object uses conditional code to find in what context it was called. If subType was passed into the method

new() of the Person object, it is being called by a subtype and should execute the instructions above.

To see if it works, create Tester6.cfm and place this code in it:

```
<cfset tally = CreateObject( 'component', 'Employee' )>
<cfset tally.new( 'Marketing', 'March 14, 2002','Tally', 'Ho',
'Female', 'June 31, 1900' )>
<cfdump var="#tally#">
```

Running this code should produce something like the screen shown in Figure 7.

We now have an Employee object that has called super.new() to use a method in Person that was overridden – new(). Hopefully, in a future release, the super concept will be directly supported in the language.

We can extend the concept of specialization even further. Employee is a broad class. We might wish to break it down even further – for example, to differentiate between salaried workers, hourly workers, and contract workers. That might translate into three new CFCs, SalariedWorker, ContractWorker, and HourlyWorker – all subtypes of Employee. Let's start by building a SalariedWorker.

Since their salary is the most distinguishing difference between a salaried worker and other workers, we will want salary as a property of SalariedWorker. Our salaried workers have bonus plans available. In our example those plans are given letter names, "A" through "D". To keep track of this, we'll add bonusPlan as a property of a SalariedWorker. Finally, with the many recent scandals concerning upper-level executives, we should probably start making allowances for the inevitable: let's make underIndictment a Boolean value. The code for SalariedWorker.cfc is in Listing 6.

To test our code this time, rather than writing more Tester code, let's create a form, SalariedWorkerForm.cfm, that lets us enter the information for a new salaried worker. While it's possible to call a CFC directly from a form, by setting the form action attribute equal to the desired CFC, I don't want such a tight link between an employee form and the CFC. For this test, then, let's set the form action attribute of the form page equal to ProcessForm.cfm. The full code for SalariedWorkerForm.cfm is in Listing 7.

The processing page, ProcessForm.cfm is very simple:

```
<cfinvoke
component="com.techspedition.CFCs.SalariedWorker"
method="new"
argumentcollection="#form#"
returnvariable="aSalariedWorker">
<cfdump var="#aSalariedWorker#">
```

Note the use of the argumentcollection attribute that neatly takes a structure and passes it along to the CFC. Run the form, fill in the information, and you should see something like the screen shown in Figure 8.

We've succeeded in creating a SalariedWorker CFC that extends an Employee CFC that extends a Person CFC – and in so doing have learned more about how to use specialization to model parent-child relationships that show up so frequently in the world.

Specialization is only one way that CFCs can relate to each other, but it is a particularly powerful means of avoiding duplication of code by abstracting common functionality from a descendant object to an ancestor. Until now, the value of inheritance has been reserved only for OO languages. With ColdFusion MX and CFCs, that changes.

About the Author

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



HAL@FUSEBOX.ORG

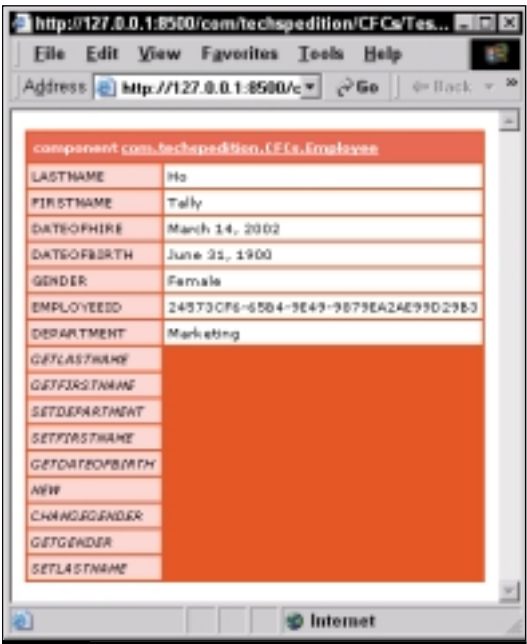


FIGURE 7 Calling the revised Employee.new()



FIGURE 8 ProcessForm.cfm

Undocumented ColdFusion MX 1.0

BY
BEN
FORTA



Discover the thrill of sailing in uncharted waters

Discovering undocumented features in your favorite application is always something of a thrill. This is especially true when those features expose little nuggets of functionality that you can leverage in your own code. And so, as promised in my last column, I'm going to introduce you to the *factory*, a set of internal ColdFusion MX services exposed via Java objects and APIs.

The Factory

When most of us think of ColdFusion we instinctively think of CFML, the ColdFusion Markup Language. This is understandable: after all, it's CFML that we spend most of our time with.

But ColdFusion is actually much more. It's also a collection of application services and runtime services, and has to be. When you build a dynamic SQL query statement something must process that and build the query. When you execute queries something needs to track that information in order to display debug output. When you use <CFCHART> an entire charting engine must be present to execute your request. The list goes on and on.

Caveat emptor:

Undocumented means unsupported – there are no examples, no documentation, and no support. In addition, any undocumented feature may be changed or removed in future versions of products. That's what distinguishes features from undocumented features; vendors are under no obligation to support anything undocumented. Use at your own risk.

ColdFusion 5 (and earlier) never exposed these core services. They were built into ColdFusion and were for the most part inaccessible. ColdFusion MX changes this quite a bit. Built on underlying Java code and Java openness, CFMX includes all sorts of services that perform all sorts of processing (starting with the examples I just mentioned).

For example, consider debugging. Ever since ColdFusion 1.0 days, debug output has been appended to the bottom of generated pages. CFMX still generates the same debug output, but it also offers a new DHTML-based dockable debug screen (which is much easier to use). If you've ever dug around in the ColdFusion directory structure, you'd have found a directory named debug (under CFMX's WEB-INF directory). In it are three CFM files:

- **classic.cfm:** ColdFusion code that emulates classic ColdFusion debug output, appended to the bottom of each page
- **dockable.cfm:** ColdFusion code that generates the new DHTML-based debugging
- **dreamweaver.cfm:** Used by Dreamweaver MX to obtain debug content for displaying in Dreamweaver's own Debug tab

In other words, unlike CF5 and earlier, CFMX doesn't automatically append debug information to pages. Rather, it collects that information and exposes it via APIs so that regular CFML code can be used to render it as needed. The debugging appended to the bottom of your generated code is placed there by CFML code that emulates the behavior of earlier versions of ColdFusion. You could create your own debug out-

put formats (modeled on the ones in the debug directory) and then save them in this directory so that the ColdFusion Administrator could see them and make them available as debugging formats.

And talking about the ColdFusion Administrator, it too is good old CFML. How does it get to validate datasources, set SMTP servers, and more? Answer: it uses APIs exposed by ColdFusion and makes calls to them.

Which brings us back to the factory and integrated services. The factory is a set of objects that expose all sorts of integrated services. It's used by the just-mentioned debug pages, as well as by the ColdFusion Administrator. You can use it too.

A Visit to the Factory

Intrigued? You should be. This starts to be lots of fun very quickly. The factory is a set of Java objects, so interaction with it is via the <CF OBJECT> tag (or the CreateObject() function if using <CFSCRIPT>). Look at this code:

```
<!-- Access factory -->
<CFOBJECT ACTION="CREATE"
    TYPE="JAVA"
    CLASS="coldfusion.
server.ServiceFactory"
    NAME="factory">
```

This <CFOBJECT> creates a local object (a ColdFusion object) named *factory* that can then be used to access factory services. You can execute the foregoing line of code as is. It will work (in CFMX, not in earlier versions), but won't actually do anything with the instantiated object. To see what's inside the factory, you can simply <CFDUMP> the object like this:

MACROMEDIA

www.macromedia.com/go/cfdjdevcon2002


```
<!-- Access factory -->
<CFOBJECT ACTION="CREATE"
  TYPE="JAVA"
  CLASS="coldfusion.
    server.ServiceFactory"
  NAME="factory">
<!-- Dump factory contents -->
<CFDUMP VAR="#factory#">
```

Execute this code and you'll see a dump of a complex nested structure containing objects, methods, and more. Everything you see on your screen is accessible to you (if you can work out what they do, what parameters they expect, and what assumptions they make – hey, that's what makes this fun!).

Accessing Factory Services

Now that you know how to access the factory, the next thing is to learn how to access specific services. Look at the top-level objects in the previous <CFDUMP>. You'll see names like DataSourceService and DebuggingService. To access any particular service just use the appropriate get method. For example, the getDataSourceService() method returns the DataSourceService, and getDebuggingService() returns the DebuggingService. To access the DebuggingService you could do the following:

```
<!-- Get "factory" -->
<CFOBJECT ACTION="CREATE"
  TYPE="JAVA"
  CLASS="coldfusion.
    server.ServiceFactory"
  NAME="factory">
<!-- Get datasource service -->
<CFSET dsService=factory.
  getDataSourceService()>
```

dsService then provides access to the DataSourceService itself. You can <CFDUMP VAR="#diservice#"> to see what it contains (it'll be a subset of the prior <CFDUMP>).

Using the Datasource Service

For our first example we'll use the datasource service. Suppose you want to programmatically validate that a datasource was valid. There's no CFML function to do this, but it must be doable. After all, the ColdFusion Administrator does it! Well, it is indeed doable, using the factory (just as the ColdFusion Administrator does).

Listing 1 is an example I used last month when discussing UDFs, and as promised, the following is the explanation of how it works.

VerifyDSN() does just that: you pass it the name of a datasource and it'll return true if that datasource can be verified (as usable) or false if not. VerifyDSN() accepts one required parameter – the name of the datasource to be verified. Let's walk through the code together.

The function and required argument are defined using <CFFUNCTION> and <CFARGUMENT>. (These were explained in the August column [Vol. 4, issue 8]).

Next, two local variables are defined: dsService will contain the datasource service object, and result is the return value (which is initialized to true – kind of innocent until proved guilty).

Then a try/catch block is defined. One of the things you'll learn very quickly about the methods exposed by the factory is that they don't throw pretty error messages or let you know when you messed up passing an argument. You need to use them exactly as intended or they will throw error messages. More on this in a moment.

Next comes the factory access code I showed you earlier.

The actual DSN validation is performed by the verifyDataSource() method, which takes the name of a datasource as a parameter and returns true if it can be validated and false if not. Whatever verifyDataSource() returns is saved to the result variable.

verifyDataSource() requires that a valid datasource name be provided; if the name doesn't exist, an error will be thrown. There are methods that return a list of datasources, so the function could first have done that and checked the name to see if it was valid, but why bother? We already trap errors with try/catch handling. If verifyDataSource() throws any error at all (including an error because the specified datasource didn't exist), we'll just treat it as a false, which is exactly what the <CFCATCH> block does.

And finally, <CFRETURN> returns result, which will be true if the datasource can be validated, or false if not (either because the validation failed or because an error was thrown).

VerifyDSN() can now be used just like any other function:

```
<CFOUTPUT>
#VerifyDSN("exampleapps")#
</CFOUTPUT>
```

If you do indeed want to obtain a list of datasources, the getDatasources() method can help. Listing 2 is an example.



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

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

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

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

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

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

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

web services
web designers
Stephen Kilmurray stephen@sys-con.com
Christopher Croce chris@sys-con.com
Catalin Stancesco catalin@sys-con.com

online editor
Lin Goetz lin@sys-con.com

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

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

This time getDatasources() is used to obtain a full datasource list. This method returns a complex structure, the top level of which contains the datasource names, and beneath are complete datasource details. As all we want are the names, the Struct-KeyToArray() function is used to re-turn an array containing the top-level key names – in other words, an array of datasource names. The names are then dumped in a simple HTML unordered list.

Using the Debugging Service

Of particular interest is the debugging service, the service used by the debugging engine in ColdFusion. This is what drives the debug templates referred to previously.

The debugging service doesn't generate any output. Rather, it keeps all information that may be of use in debugging in a query in memory. The code in the debug templates extracts data from that query and presents it as needed. And if that code can access the query, well, so can you.

A function I've always wanted was one that could return the exact

SQL statement as submitted to a datasource post any dynamic processing. ColdFusion obviously knows this information – after all, it's in the debug output and always has been, but there's been no way to access it. Execution time, record count, column list – those are all accessible, but not the SQL itself.

Well, if the debugger knows this information, and in CFMX your code can know everything that the debugger knows...see where I'm going? Take a look at the code in Listing 3.

QueryGetSQL() takes the name of a previously executed query and returns its SQL statement. Let's walk through the code.

The function and argument are defined and variables are then initialized.

The debugging service collects information only if debugging is needed (which actually makes perfect sense). This means that this function can be used only if debugging is enabled, so the processing is enclosed within a <CFIF IsDebugMode()> condition.

The debugging service is then accessed just as we did earlier with

the datasource service. The debugging information itself is stored in a query, as already explained, and the getData() method exposes that query. Thus the following snippet creates a local query named events that contains the debug information:

```
<!-- Get the event table -->
<CFSET events=
  cfd.getDebugger().getData()>
```

The events query contains all sorts of information (do a <CFDUMP> – you may find something intriguing or useful). So how to extract the single column we need for the one desired entry? We can query that query:

```
<!-- Get SQL statement (body)
for query -->
<CFQUERY DBTYPE="query"
  NAME="getquery"
  DEBUG="false">
  SELECT body
  FROM events
  WHERE type='SqlQuery'
  AND name='#queryname#'
</CFQUERY>
```

PAPERTHIN

www.paperthin.com

ABOUT THE AUTHOR

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

Each entry in the events query has a type. The type we want is `SqlQuery` (other types contain other debug information). The `WHERE` clause filters this type for the specified query name – this way we obtain just the row we want. Rows are made up of lots of columns, including one named `body` which (when type is `SqlQuery`) contains the SQL statement. That's the one column we want.

The function then saves that body column to *result* and returns the *value*. Simple as that.

Using the Runtime Service

One of the most intriguing (and, when you look at the method names, scariest) services is the runtime service. I'm still working my way through this one, but here's one method that's already finding its way into my applications.


Ever wondered how you could easily determine if the `SESSION` scope was safe to use (i.e., session state management was enabled)? The runtime service contains a method named `getVariables()` that returns a structure containing two other structures, *application* and *session*. Each of these structures contains (among other things) a member named *enable*, which will be true if enabled and false if not. I'm not going to give you the code to access these – try to work it out yourself.

Ah, the possibilities are mind-boggling!

Summary

ColdFusion MX is built on a brand-new, completely redesigned architecture – you already know that. But what you may not have known is just how open and flexible this architecture is. The factory is a perfect example of this openness, and while its use is undoc-

umented, it's also perfectly legal. Want to create an administrative program of your own? It's doable. Want access to internal data? That's doable too. Want to create graphs, work with log files, tweak the schedule, or read and write CF Admin settings? All that is doable. Want to create your own debug screens? As already explained, even that is doable.

When you tinker at this level, of course, you're on your own (never experiment on production servers!). There's no documentation on this stuff. It's undocumented – anything I've shared here was learned by browsing files and experimenting – but that's just part of the fun. So give it a try, and if you create anything really cool I'd love to see it (or submit it to www.cfml.org if you can turn it into a function). Enjoy! 

 BEN@FORTA.COM

Listing 1

```
<!--- Verify datasource is working --->
<CFFUNCTION NAME="VerifyDSN" RETURNTYPE="boolean">

  <CFARGUMENT NAME="dsn"
    TYPE="string"
    REQUIRED="yes">

<!--- Init local variables --->
<CFSET VAR dsService="">
<CFSET VAR result="true">

<!--- Try/catch block --->
<CFTRY>
  <!--- Get "factory" --->
  <CFOBJECT ACTION="CREATE"
    TYPE="JAVA"
    CLASS="coldfusion.server.ServiceFactory"
    NAME="factory">

  <!--- Get datasource service --->
  <CFSET dsService=factory.getDataSourceService()>
  <!--- Validate DSN --->
  <CFSET result=dsService.verifyDatasource(dsn)>

  <!--- If any error, return FALSE --->
  <CFCATCH TYPE="any">
    <CFSET result="false">
  </CFCATCH>
</CFTRY>

<CFRETURN result>

</CFFUNCTION>
```

Listing 2

```
<!-- Get "factory" -->
<CFOBJECT ACTION="CREATE"
           TYPE="JAVA"
           CLASS="coldfusion.server.ServiceFactory"
           NAME="factory">

<!-- Get datasource service -->
<CFSET dsService=factory.getDataSourceService()>
<!-- Get datasources -->
<CFSET dsFull=dsService.getDatasources()>
<!-- Extract names into an array -->
<CFSET dsNames=StructKeyArray(dsFull)>

<!-- List names -->
<UL>
<CFLOOP INDEX="i"
         FROM="1"
         TO="#ArrayLen(dsNames)#">
```

```

<!-- Display name -->
<CFOUTPUT>
<LI>#dsNames[i]#</LI>
</CFOUTPUT>
</CFLOOP>
</UL>

```

Listing 3

```
<!--- Get a query's SQL statement --->
<CFFUNCTION NAME="QueryGetSQL"
    RETURNTYPE="string">

    <!--- Query name is required --->
    <CFARGUMENT NAME="queryname"
        TYPE="string"
        REQUIRED="yes">

    <!--- Initialize variables --->
    <CFSET VAR result="">
    <CFSET VAR cfd="">
    <CFSET VAR events="">

    <!--- Requires debug mode --->
    <CFIF IsDebugMode()>

        <!--- Use debugging service --->
        <CFOBJECT ACTION="CREATE"
            TYPE="JAVA"
            CLASS="coldfusion.server.ServiceFactory"
            NAME="factory">
        <CFSET cfd=factory.getDebuggingService()>

        <!--- Get the event table --->
        <CFSET events=cfd.getDebugger().getData()>

        <!--- Get SQL statement (body) for query --->
        <CFQUERY DBTYPE="query"
            NAME="getquery"
            DEBUG="false">

            SELECT body
            FROM events
            WHERE type='SqlQuery' AND name='#queryname#'
        </CFQUERY>

        <!--- Save result --->
        <CFSET result=getquery.body>

    </CFIF>

    <!--- Return string --->
    <CFRETURN result>

</CFFUNCTION>
```

```
result=getquery.body>

return string --->
result>
```

CODE LISTING
 DDDDDDDDDDDDDDDDD

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

TERATECH INC.

http://www.cfconf.org/cf_underground4/

Managing Your Career in Tough Times



BY
HAL
HELMS

The wisdom of planning for the future

Ah, for the good old days! Only a couple of years ago, one of the biggest problems for developers was the constant blitz of recruiters...

...haranguing us with offers of better jobs for more money. Where are they now that we need them?

There's no doubt that today things are very different. During the heyday of the dot-com explosion, employers spent with abandon. Gripped by a mass mania, the sky was seen to be the only limit. Everyone knew that the old rules were no longer valid; we were in a new economy – one that didn't require the old, stuffy rules of actually making money. Eyeballs were the new currency of the realm.

But things are different now. The stock market, presumed to be guided by an unseen force of nature, always up and to the right, has found gravity. The go-go mentality of the dot-com explosion has been traded for fear and loathing. An old saying has it that "everyone is a genius in a bull market." That's true for individual investors and for companies. But revelations like Enron and WorldCom show us that the exuberant irrationality of the good old days was built on unreality, and in some cases fraud.

If it's any comfort, companies still employ the same, tired rhetoric: "People are our most important asset" is still boilerplate for annual reports. That may be true – if by "people" the companies are referring to top executive "people" or board member "people." For the rank-and-file employee, things are different; belt-tightening is the theme.

Both the manic and depressive phases of this cycle mark a sickness in the corporate body. The good news is that, eventually, health and sanity will return to bipolar corporate boardrooms. The bad news is that word *eventually*. Who knows how long it will take? And in the meantime, what do we developers do?

There Is Something...

On a macro level, we can do very little. We're considerably farther down the food chain than the corporate executives whose vision steered us to this point. In many cases we may never have even met those in charge. But there *is* something we can do, something that is radical and will have long-term effects for our careers: we can change the way we think about our companies. The day of relying on employers for our careers is over – if such a day ever really existed.

A friend of mine likes to remind me, "The only difference between you and the French fry cook at Burger King is what's inside your head." That's true with all so-called knowledge workers, of course, but particularly so with programmers whose current knowledge stays current only for a few months before it begins to lose its value. Economists have a term for this: a *wasting asset*.

Stated differently, our careers and livelihoods depend on our knowledge, yet when times turn tough, one of the first things to be cut from corporate budgets is training for employees. Maybe what all those well-meaning annual reports really should say is that "people are our most *replaceable* asset."

This doesn't mean that we need to be at the mercy of corporate bean-counters, though. In particular, ColdFusion MX opens up new possibilities to learn and to increase our value in the marketplace. But learning in the sense I'm talking about means more than simply studying the documentation, as important as this is. Take, for example, the issue of ColdFusion components (or CFCs), a new feature of ColdFusion MX. As ColdFusion developers come to

terms with the technology of objects in the form of CFCs, there's a good deal of talk about "design patterns." But what are design patterns? How did they come to be – and how can this help us guide our own careers in these tough times?

Design patterns originate from the world of architecture. Christopher Alexander, mathematician, philosopher, and architect, organized ways people solved recurring problems in building and called these *patterns*. One such pattern is called "nine percent parking." Here's what Alexander says:

The integrity of local transport areas and the tranquility of local communities and neighborhoods depend very much on the amount of parking they provide. The more parking they provide, the less possible it will be to maintain these patterns, because the parking spaces will attract cars, which in turn violate the local transport areas and neighborhoods.

Very simply, when the area devoted to parking is too great, it destroys the land. Therefore, do not allow more than 9 percent of the land in any given area to be used for parking.

Once successful patterns such as this are determined, architects and planners are able to use these patterns as models and rules. They are not forced to reason from so-called "first principles."

But how did we get from nine percent parking to computer program designs? Good developers know syntax; they understand data types and algorithms. Great developers do all this but go beyond: they draw unexpected connections between seemingly dissimilar things and events. Design patterns are one such unexpected connection.

A group of programmers who were exposed to Alexander's ideas began making connections between the problems faced by urban planners and those faced by computer programmers. Could patterns be applied not only to their original domain, architecture, but to program design as well?

In 1995 Eric Gamma, Richard Helm, Ralph Johnson, and John Vlissides (who later came to be known as the "Gang of Four") wrote a book called *Design Patterns* that applied the work of Alexander in identifying patterns to problems encountered by computer programmers.

Today, unfortunately, the term *design pattern* has become an official buzzword, a designation that all too often closes the subject to further thinking and discussion. Yet that's not the fault either of Alexander and his original work or of the connections so brilliantly made by the Gang of Four.

It's safe to say that Gamma et al. did not come upon their knowledge by attending a class on design patterns. Nor did any pointy-haired boss of theirs say to them one day, "Boys, I want you to look into how problems are solved by architects. I bet there's something to be learned there." Instead, they relied on their own curiosity, passion, intelligence, and intuition as guides.

As someone who trains other developers, I certainly don't mean to impugn the value of classes. I think such classes, and the personal mentoring they can provide, are immensely valuable. In fact, if your company is one of the forward-thinking ones that's actually investing in its programmers, I have an idea or two about classes you might want to look into!

Re-create Your Own Careers

But what are we to do if those in charge of the budget have decided that training is low-hanging fruit, easy picking in the search to slash costs? Are we to let our knowledge wither while we wait and hope for a change in the economic climate? That's certainly the path of least resistance, but if we follow it we'll pay the price when our knowledge becomes obsolete. The loyalty that executives often talk about seems strangely to flow in one direction only – upwards – and if we don't take responsibility for our own training we'll be left behind.

Mark Twain once remarked, "Everyone complains about the weather, but no one does anything about it." Let's do something other than waiting for the economic weather to change. I'm beginning by offering free online training on my Web site, www.halhelms.com. Each month I'll have a new lesson on an aspect of programming, complete with a self-test to see how you fared.

I'll also have a recommended book for that month, something to encourage you to study further so you can make your own unexpected connections. What the economy will do is anyone's guess, as is how your company will respond to it.

Regardless of what happens, we can take responsibility for our own careers, and in this we can do no better than to heed the advice offered by Alan Kay, the father of Smalltalk: "Don't worry about what anybody else is going to do. The best way to predict the future is to invent it!"



HAL@FUSEBOX.ORG

ABOUT THE AUTHOR

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

HOSTMYSITE.COM

www.hostmysite.com

An Insider's View

BY
JIM
SCHLEY



Tips for a trouble-free migration to ColdFusion MX

Migrating your application to ColdFusion MX from a previous version is a project that's probably high on many of your priority lists.

Macromedia, like many other companies, has a multitude of disparate internal and external Web sites that are powered by ColdFusion. The week before CFMX CDs were sent out to the manufacturing plant for duplication, we took the opportunity to upgrade many of our systems to CFMX. Additionally, throughout the beta cycle, I worked with many of you personally to help test your applications on CFMX.

We learned a lot from these two exercises, and much of what we learned helped result in a product designed to be highly backward compatible with previous versions. Despite CFMX's being rewritten from scratch and including a host of new features, a huge effort was put into providing a high degree of CFML language support from previous versions and tools to analyze your code for deprecated and obsolete features.

Since CFMX's release, I've spoken with many of you directly or indirectly about your experiences in migrating to the new product. In this article I'll share with you what's been shared with me and what I know about CFMX so you may have a trouble-free migration.

Analysis and Testing

Performing a CFMX upgrade is not unlike performing any other software upgrade on a server system. To minimize impact on the end users of your application, a well-thought-out migration plan in is order. I cover the topic of creating a migration test plan in more detail in my article at the ColdFusion MX Application Developer Center (www.macromedia.com). The central concept, however, is that testing your code on CFMX before you upgrade your production server is absolutely essential.

Some applications can be upgraded without any code modification at all. Others, those that use CFML features no longer supported, must first be modified. Your first step in identifying potential incompatibilities is to run the Code Compatibility Analyzer available in the ColdFusion Administrator. This utility allows you to select a directory and scan all code in it and its subdirectories. The tool detects instances of deprecated or obsolete CFML in your code and produces a laundry list for you to follow.

Once you've made any necessary code changes in your development environment, it's time to upgrade your QA/Test/Staging environment to CFMX and begin testing. I generally recommend using whatever testing procedures you normally use when modifying application code to test that your application is running smoothly in the new environment.

Making sure that your code runs on CFMX without any errors is the first step. To gain a high level of comfort that your application is going to run smoothly in production, I suggest you run a realistic load test against it using any software package that's capable of simulating the stress your server typically experiences. This type of test can help you find potential problems before there's any chance your end users could run into them.

Compatibility

As I mentioned above, it may be necessary to make modifications to your code because of obsolete or deprecated CFML in your code. The CFMX documentation includes a complete list of changes to the supported CFML in the *Migrating ColdFusion 5 Applications* manual (http://livedocs.macromedia.com/cfm5/1docs/MigratingColdFusion_5_Applications/contents.htm). The Code

Compatibility Analyzer, as mentioned above, is your shortcut to finding these issues in your code.

One item that seems to come up in code that isn't picked up by the Code Compatibility Analyzer is a change to the list of reserved words on CFMX. Reserved words can't be used as variable names in CFML. All the names of variable scopes have been added to this list of reserved words with this release and should no longer be used. Unfortunately, some of these names are commonly used by developers because of their simple, descriptive nature. To give a few examples, variables can no longer be named things like url, file, application, etc. There is a full list of reserved words in the CFMX documentation.

One less minor change to CFML is the overhaul of the Advanced Security service. CFMX no longer contains an OEM version of Netegrity SiteMinder as previous versions did. Instead, security is implemented with the native Java security services. As a result, the old Advanced Security tags such as CFAUTHENTICATE are no longer available. The new security framework has all new tags to use, such as CFLOGIN. If you wrote an application that made use of Advanced Security policies and rules in an earlier version of ColdFusion, you may need to do some significant rearchitecting.

I'd hazard a guess that this isn't the case for most people, although I know many people (including me) used CFAUTHENTICATE to authenticate against Windows domains. CFMX no longer has native capability to authenticate users against a Windows domain, so calls to CFAUTHENTICATE for this purpose will have to be replaced. Fortunately, the change is simple, thanks to a module that ships with JRun 4 for Windows and is available to CFMX users if you know how

to get at it. The following code can be used to replace CFAUTHENTICATE:

```
<cfscript>
NTAuthObject =
createObject("java", "jrun.
security.NTAuth");
NTAuthObject.
init("#form.domain#");
NTAuthObject.authenticateUser("#
form.username#", "#form.
password");
</cfscript>
```

This code will error if the username and password arguments to the authenticateUser method aren't correct for the domain passed to the init method, so error handling is a must here. For more information on using JRun's Windows authentication module in ColdFusion, see Rob Rusher's article, "Security Best Practices: Authenticating and Authorizing Against NT Domains with CFMX," in the CFMX Application Developer Center.

I'd like to emphasize that CFMX is highly backward compatible, and code changes required for most

users are minor or not required at all. The important concept here is that you find out if your code is ready for CFMX before you upgrade.

Architecture

It's important to remember that upgrading to CFMX means switching to a completely new application server platform. J2EE application servers are designed to be platform neutral, meaning they run just as well on Windows as they do on Solaris or Linux. This is a major benefit to moving to this new platform; you can expect the same stability and high reliability regardless of what operating system platform you choose. Many other benefits are associated with running on a J2EE-compliant application server, but one potential downside is that platform-specific application code may not run as optimally or may run differently.

A specific example of this is ColdFusion's access to COM objects on Windows. CFMX continues to provide convenient programmatic access to COM objects in CFMX via the CreateObject function and the CFOB-

JECT tag. This ColdFusion-COM interaction is unchanged in CFML; however, the underlying architecture to access COM objects is completely changed in CFMX. Earlier versions of ColdFusion could natively access COM objects internally because the server was written in C++. Since CFMX is running on top of a J2EE application server and its process is isolated from the system by the Java Virtual Machine (JVM), COM objects need to be accessed via the Java Native Interface (JNI). This adds a level of complexity to the overall operation, and although this complexity should be abstracted from end users, some issues have come up and are documented in a Macromedia TechNote titled "Issues with Using COMObjects in CFMX."

Another example of this is with ODBC data access on Windows. In previous versions of ColdFusion, ODBC was one of the primary ways to communicate with a database in CFML. Since CFMX runs in the J2EE world, JDBC supplants ODBC as the primary database communication layer. In almost all cases, when talk-

CFDYNAMICS
www.cfdynamics.com

ing to an RDBMS system that Macromedia ships a JDBC driver for, your existing database code will work just fine. The only exception would be if you wrote some ODBC-specific API calls inside your queries. You know who you are if you did this, and you need to change your code for JDBC compatibility.

Some databases and applications require ODBC to communicate to them. Microsoft Access is a prime example. CFMX ships with a Type III JDBC driver and middleware server OEM'd from DataDirect that allows you to connect to your ODBC datasources, including Microsoft Access. You can learn more about this middleware product and Type III JDBC driver at DataDirect's Web site. Our experience shows that the stability and performance of the ODBC middleware server is more sensitive to underlying problems in the ODBC layer than previous versions of ColdFusion. Because of this, we highly recommend upgrading your Microsoft Data Access Components (MDAC) package to the latest released service pack, that is, MDAC 2.6 SP2 or higher. MDAC updates are available for download from Microsoft.

In any production environment where your site is stressed to any degree, I strongly suggest that you use an RDBMS system that's designed for high performance. Additionally, use a Type IV JDBC driver for the best performance. As mentioned above, make sure to test your system, including the database and JDBC connection, under load before placing it into production.

Memory

With regard to memory, CFMX and the JRun J2EE application server behave quite differently from previous versions of ColdFusion. This is primarily due to the fact that the server now runs entirely within the confines of an instance of a JVM. The mechanics of a JVM are beyond the scope of this article, but the concept of a memory heap in the context of a JVM is important. When a JVM starts up, it immediately grabs a set amount of memory to use – this is called a *heap*. This

startup size is quantified by the “min heap size” setting and is configurable via the CFMX Administrator on the “Java and JVM” page as the “Initial Memory Size.” The amount of memory that a JVM is able to use is limited by the “max heap size.” This is also configurable via the CFMX Administrator as the “Maximum Memory Size.” Min and max heap-size terminology is the Java vernacular, whereas the ColdFusion Administrator uses more descriptive terms – they're interchangeable. These settings are very important to the proper operation of a Java application server, and although they may be new to some ColdFusion system administrators, their importance should be highlighted.



For maximum performance, stability, and reliability, the max heap size should be set to a value sufficiently large so the memory needs of CFMX can be met by memory available to the JVM, but not so large that the JVM has to access operating system virtual memory or swap memory. You need to choose a value that is reasonable based on your machine's hardware.

For instance, if you have a server with 512MB of memory, it would be advisable to limit your max heap size to around 384MB. Since the operating system and other running applications each make a grab for their own memory, you realistically won't have much more than 384MB left over for the JVM to make use of.

Another reason why load testing your application is advised is that it's really the only way you can get an idea of exactly how much memory your application requires to run optimally. This is done by watching memory use by the JVM process while your load test is in progress.

This memory usage will be much higher than when you hit the application with a single user's load because of the concurrency. If you see that your memory use is at or near the max heap size, you won't be performing optimally and should consider increasing your heap size if possible. The optimal max heap size may be larger than the memory footprint of a previous ColdFusion version's server process. These memory requirements are important to know about before upgrading your production server.

There are a lot of different factors within your application code that affect your overall memory requirements, but cached queries seem to be a common memory hog (more so in CFMX than in previous versions). If your application uses a lot of cached queries and you find your memory footprint to be large or you receive “out of memory” errors from your JVM, you should consider reducing the cached query setting in the ColdFusion Administrator.

Java Virtual Machines

The last topic I should mention that might be new to many ColdFusion users is that of JVM selection. Nothing is more vital to the performance, reliability, and stability of any Java application server than the underlying JVM. CFMX ships with Sun Microsystems' 1.3.1_03 JVM. This particular JVM was found to be highly reliable in our QA testing, and that was the primary concern. For optimal performance or other reasons, you may choose an alternate JVM, such as the newer 1.3.1_04 or 1.4.0_01, for instance. As with implementing CFMX, implementing a new JVM is a significant change that requires significant testing. Other JVMs are available from other vendors, such as IBM and BEA, and can be used with CFMX if you have a license from the vendor for their use. Again, the importance of testing here can't be emphasized enough.

So, with this info in hand, I hope your CFMX upgrades will be all the easier to get checked off your punch list.



@ JSCHLEY@MACROMEDIA.COM

MACROMEDIA

www.macromedia.com/go/cfmxmlight

ABOUT THE AUTHOR

Jim Schley, a QA manager for ColdFusion at Macromedia, worked as principal QA engineer for the ColdFusion MX release. He has six years' experience building Web applications and has been programming with ColdFusion since version 2.0. In addition to writing his own articles on ColdFusion, Jim has worked as a technical editor and consultant on several popular ColdFusion books.

Hidden Gems in CFMX

BY
CHARLES
AREHART



Mining the available resources

With each new release of CFMX some features are trumpeted a lot and others seem to fall between the cracks. I call these “hidden gems,” and in CFMX there are quite a few.

In this article I’ll uncover several features that don’t warrant publicity in the popular top 10 (or even top 20) lists that Macromedia and others might put out. Perhaps they won’t influence the average IT buyer considering an upgrade. Sure, things like J2EE integration and XML/Web services support are critical to many organizations, and things like CFCs and Flash remoting may fundamentally change the way we develop CF apps.

But it’s been my experience that what really intrigues most developers is the little things that have changed, some of which just may serve to make every day of our development lives easier without any substantial changes in current practices.

The big new things in CFMX are of course praiseworthy, and I don’t mean to diminish them. It’s just that maybe for some developers one or more of these hidden gems could be the real “gotta have” trigger that motivates them to upgrade. Or solve a long-standing problem.

I’ve broken the topics into a couple of categories. I’m not even mentioning all the hidden gems, nor even all the important changes to CFMX. As such, this article shouldn’t be considered a compendium of all hidden gems. I can only highlight several.

Not-So-Hidden Gems

Before proceeding, I want to point out that there are additional new features of CFMX that may not get quite as much widespread acclaim. I don’t consider them to be hidden gems since they’re talked

about often and are well documented. If you haven’t yet looked into them, you should:

- **CFFUNCTION:** Creates UDFs that can include tags
- **CFLOGIN, related tags:** Support standardized login processing
- **CFIMPORT:** Enables JSP custom tags
- **CFCHART:** Replaces and exceeds CF5’s CFGGRAPH
- **Automatic compilation of CF templates**
- **Support for internationalization (UNICODE)**
- **Improved debugging** with CF-TRACE

There are others, but you get my point. There’s a lot more to this new release than just the “top” few features that are so often discussed.

At the end of this article I’ll point out several resources available for learning more about these and the other features I’ll mention.

Hidden Gems in CFML

I’ve grouped the first set of hidden gems in a general category of CFML (ColdFusion Markup Language) gems, meaning new or changed tags and functions.

New Variable Scopes

More and more CF variable scopes have become available to use as structures. CFMX adds to this list the variables, caller, and server scopes. The value here is that you can use the CFLOOP COLLECTION tag to loop over the values in these scopes, or, more simply, you can display all the variables in a given scope with CFDUMP, among other things.

For instance, if you’re not familiar with the server scope, try `<CFDUMP VAR="#server#">`. The output will show several useful variables you may never have known about, including several describing the CF server and the operating system it runs on. The server scope can also be written to, in which case any user on the entire server has shared access to the variables placed in that scope.

CFSETTING REQUESTTIMEOUT

It’s now possible to programmatically override the “request timeout” duration for a given template, using `<CFSETTING REQUESTTIMEOUT="nn">`, where “nn” is a number of seconds. The CF Administrator has a value set that controls the maximum time a CF template is allowed to run before the server “times out” the template. If you have a template that needs to run longer than that default, this is the means to programmatically indicate that it should be allowed to run longer.

Some will recognize this as an alternative to (and in some ways an improvement over) the long-existing means to make this sort of template-specific request to override the “request timeout” value. In previous releases you could add `REQUESTTIMEOUT=nn` to the query string of a URL requesting a template, and that indicated the maximum runtime for the template. That approach had its pluses and minuses, but it’s important to note that this old query string approach is no longer supported (any value provided is ignored).

Try/Catch in CFSCRIPT, UDFs

Prior to CFMX, there was no way to perform try/catch exception handling in the CFSCRIPT language (which was the means to create User-Defined Functions). This limitation has been lifted and there is now a Try and a Catch statement available in CFSCRIPT. See the documentation references provided later for more details.

Note: As opposed to CFML’s CFTRY and CFCATCH tags, in CFSCRIPT, a Catch is not embedded within a Try. Also, while there are CFTHROW and CFRETHROW tags in CFML, there are no corresponding throw and rethrow statements in CFSCRIPT.

“Importing” CF Custom Tag Libraries, “Adaptive Tags”

In my article “Using JSP Custom Tags in CFMX” (*CFDJ*, Vol. 4, issue 5), I showed how you can use the new CFIMPORT tag to allow your applications to use JSP custom tags. One thing I didn’t mention is that there’s a side benefit to CFIMPORT, even if you’re not using JSP custom tags. You can also use it to point to a directory that holds one or more CF custom

tags (any directory with a relative path or using a mapped path in the CF Administrator). With that, you can then use JSP-style custom tag syntax to refer to your CF custom tags.

For instance, if you used `<CFIMPORT TAGLIB="pathname" PREFIX="mytags">` to create a reference to some directory of tags that included `sometag.cfm`, you could now refer to it in your code with `<mytags:sometag>`.

There’s also an interesting new possibility being referred to as “server-side HTML” or “adaptive tags.” If you leave off the PREFIX in a CFIMPORT or set it to the empty string, then CF will process every tag (other than CF tags and taglib:-prefixed tags) as if it’s a custom tag. It will look in the named TAGLIB to see if there’s a template of that name, and will run it if there is. So if you had a page with such a CFIMPORT and had a file named `form.cfm` tag in such a taglib, a `<form>` tag on that page would cause CF to run `form.cfm` like a custom tag as it processed the page. Very interesting possibilities. Again, it *cannot* be used to override CF tags.

By the way, for those using CFIMPORT for JSP custom tags, I failed to mention in the article (available online at www.sys-con.com/coldfusion/article.cfm?id=435) that the feature of importing JSP custom tags is *not* supported in the Professional edition of CFMX. It’s available only in Enterprise (or the Evaluation or Developer edition, which it becomes after 30 days of trial). I should clarify, however, that the notion of using CFIMPORT to work with CF custom tags does indeed work in all editions.

Evaluations Within Expressions

This hidden gem may not excite many, and it’s actually a return to the way things were done in the past. You can now perform evaluations within expressions without needing to use the `evaluate()` function. For instance, if you wanted to display how long it took to run a query that was just executed, you couldn’t (in CF5) just say:

```
<cfoutput>Time used:
#cfquery.executiontime/1000#
seconds</cfoutput>
```

E-ZONE MEDIA
www.fusetalk.com

You'd need to use an evaluate around the expression `cfquery.executiontime/1000`. That's no longer needed. Removing it where it's not needed could yield a modest performance boost.

Miscellaneous Hidden Gems

Several other hidden gems that aren't part of CFML are worth pointing out.

CFMX Tag Updater for CF Studio

Folks who are still running CF Studio/HomeSite 4.5 or 5 will be thrilled to learn that they can arrange to install a "tag update" for the CFMX tags and functions. This way, the tag/function insight, tag editor, and tag/function help features work for most CFMX tags and functions (see www.macromedia.com/software/coldfusionstudio/productinfo/resources/tag_updaters/). Also, there's an update for the Studio/HomeSite Help References (Help>Open Help References Window), available at http://download.macromedia.com/pub/home_site/updates/cfml_ref.zip. Though the instructions in the zip suggest it works only for HomeSite + (part of the new Studio MX), the update also works in Studio/ Home-Site 4.5 or 5.

The available
Developing
ColdFusion MX
Applications with
CFML manual is a
whopping 900
printed pages!

Free CFMX Hosting (Nonproduction Basis)

If you're reluctant or unable to install CFMX, you may want to seek out a hosting company to do your testing. Fortunately, as of July, one company is offering free CFMX hosting for trial (nonproduction) purposes. See www.hostmysite.com/cfmx.

[hms](#) for signup details. If anyone learns of other such instances, please let me know, or you can simply add them to the comments area of the online version of this article (see the end of this article for more on that).

Running CFMX on Mac OS X

One topic that seem to be generating a lot of buzz among some members of the community is the question of whether CFMX can run on Mac OS X. Regardless of whether and when Macromedia will support that configuration, there are people hot to make it work, and there are at least a couple of resources where you can learn more about these efforts. The first part of an article series by CFer Dick Applebaum is available on the O'Reilly site at www.oreillynet.com/lpt/a/java-script/2002/06/21/coldfusion_one.html.

Precompiling ColdFusionMX Templates

You may know that CFMX now compiles templates into Java classes. You may also have noticed that this means the first user to run a template after it's created or changed will suffer a small hit (up to a couple of seconds) when it's compiled the first time.

You can alleviate that wait by precompiling one or more templates. Unfortunately, the steps to do this take a little more room than we have for this article, so I'll save them for next month. There's also the matter of how to compile templates outside the default `wwwroot` directory (for those that have set up their servers to support that). In the meantime you can check out my blog at <http://cfmxplus.blogspot.com/> for a preview of the topic.

Other Changes Worth Investigating

There are still more changes and/ or new features that I just don't have time to discuss. As mentioned in the next section, there are several resources from which you can learn more about these. Consider whether any of them may be of interest to you:

- Several Query of Query enhancements.
- Several CFLDAP changes.

- CFMAIL spooling option changes.
- Sort order in ListSort, ArraySort.
- Date processing functions have short, medium, long formats.
- Localization functions now follow Java standard rules.
- Several Regular Expressions support changes.
- CFTHROW can now throw an OBJECT.
- CFLOG now always writes thread, date, and time.
- CFHTTP reading text file in as query, new FIRSTROWASHEADERS attribute.
- COM object access improvements.
- Dynamic class reloading.
- New commenting flexibility.
- How to save the compiled template as .java (versus .class) files.
- Can indicate alternative directory for templates outside of normal `<cfusionmx_home>\wwwroot`.
- At least two Security Bulletins currently available.
- Many CF admin settings are now saved in `<cfusion_home>\lib\neo-runtime.xml`.
- Examples in online CFML Reference are now housed at [Macromedia.com](http://www.macromedia.com) (avoids security issues of running them locally).

One more is the new GetPageContext function. As I mentioned in my June article, "New Possibility in CFMX: Server-Side Redirects" (Vol. 4, issue 6), one of the most compelling possibilities to come out of that is allowing redirection from one CF page to another (or to a JSP or servlet) without a client-side redirect (which is what CFLOCATION does). See that article, available online at www.sys-con.com/coldfusion/article.cfm?id=450, for more information.

There are still many others. You really owe it to yourself to read the resources mentioned next to make sure there aren't changes that could concern you.

Indeed, some of these issues fall more into the area of "hidden traps" for migration. I intentionally am not writing about those aspects of the upgrade. There are ample resources for that, not the least of which is the new manual, *Migrating CF5 Applications*, available within the substantial new CFMX documentation set. See the next section for how to obtain that both in print and online.

Finding More Information

The sources of many of the features I've mentioned here are the very documents that come with the new release. The thing is, many people don't know they exist or for some reason don't bother to or don't have the time to read them. I always point them out when possible, and it's especially important for this new release.

The entire CFMX documentation set has been revamped and added to. There are four updated and five new books. If you're not aware that there is documentation, or figure the books are just for reference, you really need to check them out. So many questions people ask are answered in the manuals.

The most substantial of them, *Developing ColdFusion MX Applications with CFML*, is a whopping 900 printed pages! No need to wait for books to hit the market: Macromedia has written one, and it's been improved over previous releases. Besides covering all the new features in a user-guide (rather than reference) style, it even has

improved discussions of "old" subjects such as Query of Queries, UDFs, and CFSCRIPT.

You may find the manuals on your server at <http://<yourserver>/cfdocs/dochose.htm>. They're also available online at <http://live-docs.macromedia.com/cfmxdocs/>, and in PDF form at www.macromedia.com/support/coldfusion/documentation.html. Or you can buy them in print format (only \$50 for the set) at <http://dynamic.macromedia.com/bin/MM/store/US/product.jsp?category=/BooksVideos/Documentation/ColdFusionServer&type=FULL>.

The books do indeed cover most of the new features, but there are always hidden gems in the "Release Notes" (www.macromedia.com/support/coldfusion/releasenotes/mx/releasenotes_mx.html), the "Documentation Updates" (www.macromedia.com/v1/Handlers/index.cfm?ID=22811&Method=Full), and the "Documentation Additions" (www.macromedia.com/v1/Handlers/index.cfm?id=22993&Method=Full).

There is also substantial coverage of new features on the Macromedia Designer/Developer site (www.macromedia.com/desdev/mx/coldfusion/). And the "Upgrade FAQ" (www.macromedia.com/software/coldfusion/productinfo/faq/upgrade/) and "New Feature Comparison Grid" (www.macromedia.com/software/coldfusion/productinfo/upgrade/comparison.html) can also be useful.

Finally, don't miss the example applications that you'll likely find installed on your CFMX server, at <http://<yourserver>/cfdocs/exampleapps/index.cfm>.

• • •

There's a lot more to CFMX than it may seem at first. Indeed, I'd say "there's gold in them thar hills." Keep mining these resources for still more hidden gems, and feel free to add them to the online version of this article (whose link I'll offer on my list of articles at www.systemmanage.com).

CAREHART@SYSTEMMANAGE.COM

CFXHOSTING
www.cfxhosting.com

CREATING MAINTAINABLE WEB SITES

Implementing MVC patterns with ColdFusion components

"It doesn't matter to me that you use my methodology," said the Fusebox expert sitting across from me at lunch. "What matters is that you have a methodology." And I thought back to my early experiences with ColdFusion...

I developed my first ColdFusion application by the seat of my pants. I opened the manual, looked through the sample applications, and got started. On looking back, I was surprised that the application performed relatively well and was somewhat maintainable. I got lucky.

On my next ColdFusion application I was part of a team. We dove in and started programming. The application scope grew, and we added more code to support the new specifications. Eventually the amount of time devoted to developing new features was dwarfed by the time required to make the new code consistent with the old code. Our application became unmaintainable, and after about a year it collapsed from its own weight.

As I continued developing, and discussed application design with other CFers, I learned more about what worked and what

didn't, and came up with standard designs to improve both performance and maintainability. And as I documented these approaches, I was creating what are known as "patterns": consistent, explainable relationships between software components that improve with usage and evolve over time.

I wasn't alone. The challenge of creating consistent, maintainable code is universal to all software development, and there are thousands of articles and books describing various solutions. Allaire, and now Macromedia, historically haven't enforced a single application architecture. Instead, they've provided the tools and allowed developers to come up with their own architectural solutions. As a result, various communities and companies have created their own patterns (Fusebox, CFObjets, and Allaire's own Spectra all come to mind).

ColdFusion MX offers new tools for creating a clean, maintainable application architecture. It still doesn't enforce a single solution, but it now provides the tools required to adapt some of the patterns that are popular in object-oriented languages like Java. The most important new construct is the ColdFusion component, a file that, in the words of the ColdFusion documentation, can "encapsulate application functionality and provide a standard interface for client access to that functionality."

This article describes possible ColdFusion adaptations of a common software design pattern known as Model-View-Controller, or "MVC", and explores how to implement this pattern in CFMX while leveraging the strengths of CFCs.

This article is investigation, not prescription. I do not intend to present a complete, mature application architecture to compete with existing systems. Instead, take these as thoughts that occurred to me over the last few months while developing my first "from scratch" CFMX application.

A Maintenance Nightmare

Consider a typical ColdFusion page that has been developed by a beginning programmer. The page's purpose is to query a database table and present its contents. The original ColdFusion page would execute these steps:

- Query the database.
- Present the data in an HTML table.

This is fine, as long as the programmer wants to present the data only one way. But inevitably the specifications expand. The programmer is asked to support not only a typical Web browser, but also wireless devices such as mobile phones and PDAs. So the original page is duplicated twice, and the output code is modified to support the new devices. The developer now has three pages, each of which includes a copy of the database query (see Figure 1).

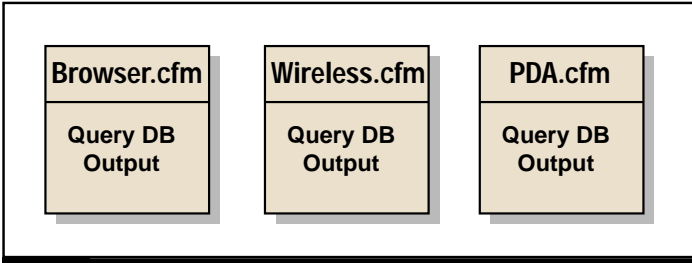


FIGURE 1 ColdFusion without components

Now the database administrator informs the programmer of a change to the database table structure; the nightmare begins. The programmer has to track down all the queries that address that table and update them. Multiply this out to a large application with thousands of pages and queries and the programmer hopes for the layoffs to begin.

Model-View-Controller

MVC was originally developed in Smalltalk as a methodology for interpreting and reacting to end-user input. It has since been adapted in the world of Java in Swing (a graphical user interface component kit) and in J2EE application servers such as JRun, WebSphere, and WebLogic as a system for developing maintainable Web applications. (For detailed information on the J2EE implementation of MVC, see http://java.sun.com/blueprints/patterns/j2ee_patterns/model_view_controller/.)

The MVC pattern separates application logic, content, and presentation, resulting in more maintainable and reusable software components. There are three primary components in an MVC implementation:

- **Controller:** Receives and interprets user input. In this CFMX adaptation this is a ColdFusion page that receives and reacts to the user request.
- **Model:** The interface to the application's data. It encapsulates queries and other server-side operations, and manages state and data caching. In this CFMX adaptation the Model is implemented as a CFC.
- **View:** Creates a text and/or graphic response for the end user. Its jobs include rendering of data models and presenting user interfaces for additional user input. In CFMX the View can be an included ColdFusion page or a custom tag.

The MVC pattern can be visualized as in Figure 2.

Implementing MVC in CFMX requires at least three files. The Controller page (user_list.cfm, see Listing 1) is called from the

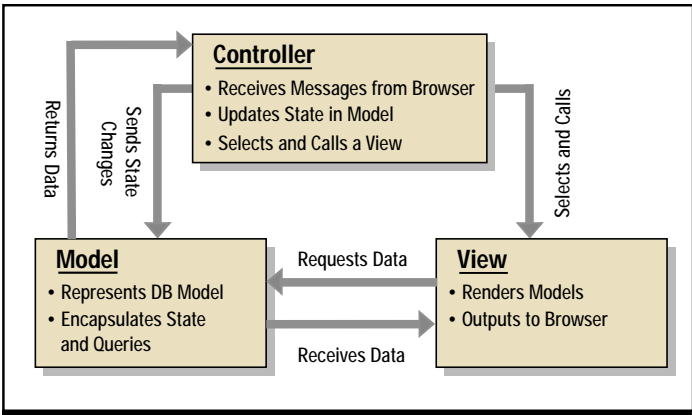


FIGURE 2 Model-View-Controller pattern

browser. The Model (user_db.cfc, see Listing 2) is a ColdFusion component that's designed to be instantiated and held in memory for some amount of time. In this implementation two CFCs have a parent-child inheritance model. The CFC in Listing 2 extends and inherits the functionality of a "parent" CFC in Listing 3. The View is an included ColdFusion page (user_list.cfm, see Listing 4) that contains nested Views (view_row.cfm, see Listing 5).

The MVC Conversation

The end user initiates a conversation by creating user input. In a Web application the input might be a click on a hyperlink or a submit button.

The Controller receives the resulting request from the browser and executes at least two steps. First it creates an instance of the Model CFC in the Request scope. Next it examines the URL scope for a "sort" argument; if it exists, it passes it to the Model, modifying the Model's current state. Finally, it calls a View that's designed to output the contents of the Model. The Controller code looks like this:

```
<cfset request.data=createobject("component", "user_db")>
<cfif isdefined("url.sort")>
    <cfset request.data.setSortOrder(url.sort)>
</cfif>
<cfinclude template="user_list_view.cfm">
```

The Model CFC is an instance of an object with properties and methods. It's designed to encapsulate and provide access to information in the database query, while protecting the data consumer (the View) from having to know anything about the data's source or structure. Figure 3 shows the output of the CFC when I browse it directly. I see the component's method names, and can tell which of the methods were inherited from the parent CFC.

When the component is first instantiated, it runs its parent CFC's constructor code (the code prior to the method definitions) and creates a locally owned variable named "this.query" as a nonquery type. When data is first requested from the object, the component sees that the query hasn't been created and runs an initialization method:

```
<cfif not isquery(this.query)>
    <cfset this.init()>
</cfif>
```

The init() method in the child CFC actually executes the query:

```
<cfquery name="this.query"
    datasource="myDSN">
    SELECT * from users
    <cfif this.sortorder neq "">
        ORDER BY #this.sortorder#
    </cfif>
</cfquery>
```

In the listing, you'll see that the query has been wrapped in a <cftry> block. If the query fails (as it will on your system, as you don't have my database), the code forks into a routine to manually create a query object with the expected fields (see Listing 6). This is an important lesson in encapsulation. The Model CFC is in charge of getting or creating the data. The View and Controller know only that data is available, but don't know where it came from.



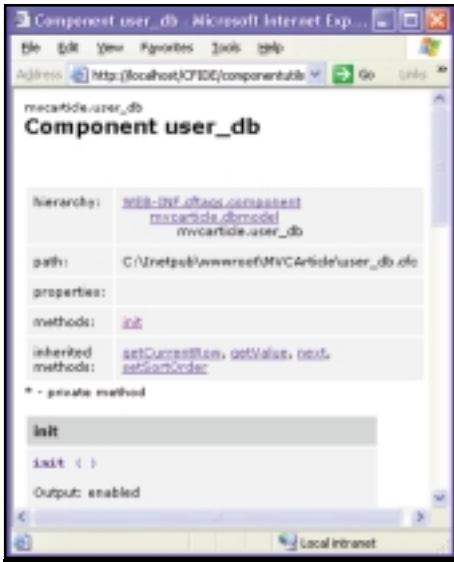


FIGURE 3 ColdFusion component metadata

The View page is included only when the CFC instance has been placed in the Request scope. The View's first job is to create a reference to the CFC instance:

```
<cfset data=request.data>
```

The View then outputs an HTML table presenting the data to the user. Notice that it doesn't use a conventional `<cfoutput>` loop with a query attribute. Instead, it uses the CFC's `next()` and `getValue()` methods to retrieve data for output:

```
<cfloop condition="data.next()">
  <cfoutput>
    <tr>

      <td>#data.getValue("user_name")#</td>

      <td>#data.getValue("user_city")#</td>
    </tr>
  </cfoutput>
</cfloop>
```

Why the change in syntax? One of the Model's most important responsibilities is to hide the structure and source of the data from the View. Remember that your system can be thrown into chaos by a simple change in data structure. What would happen in an even more extreme situation, where the data no longer comes from a database but is now retrieved from LDAP, XML, comma-delimited text, or a Web service? The Model's job is to get the data and serve it back, regardless of source or structure. The CFC's interface serves that purpose. The View doesn't know it's operating on a ColdFusion query; it knows only that

the Model object has the data and knows how to provide it.

Now let's see what happens when the specifications expand as before. When the developer is asked to support multiple devices, it's easy to develop additional View files and call them dynamically from the Controller:

```
<!-- Assumes a session.
device variable -->
<cfinclude template="user_list_#ses-
sion.device#.cfm">
```

The developer can add as many View pages as necessary. Each of the Views is maintained independently, so fixing an output bug doesn't impact the Model or Controller code. The Controller is responsible for choosing which Model and which View to use; again, changes in this logic don't impact the other two parts of the model.

If the structure or source of the data changes, the developer modifies the Model CFC. The Model is maintained independently, and the Controller and View files are left alone. As long as the data coming from the Model matches the View's expectations, the Model's internal implementation can vary as needed.

This independence of the View from the Model, and vice versa, is known as "loose coupling." They are designed to work together, but can also work with other partner components as long as they agree on the syntax of their conversation.

Views Within Views

The View is the outward presentation to the user. A single View can represent the entire page, but it's also common to have small pieces of visual presentation nested within the larger view.

A good example of this is the common CF practice of including header.cfm and footer.cfm files to give your application a common look and feel. Both the header and footer files create browser output, so they are Views. They are then nested within a "master" view. Figure 4 is a visualization of this architecture.

Each part of the nested View outputs its result directly to the HTTP response. To get data, each converses directly with the Model instance stored in the Request scope (see sidebar). The use of nested Views is a good reason to place data in the Request scope rather than passing the CFC instance directly from the Controller into the top View, and so on down the calling chain. Since all Views get their data from the same place, they're independent

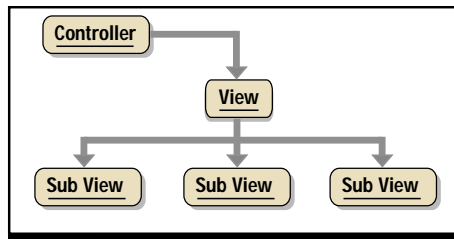


FIGURE 4 Nested Views

of each other, and modifying one doesn't have a negative effect on any other.

In this final version of the View, I've separated the section that outputs the HTML table row into its own nested View. Listing 5 is a custom tag that expects a comma-delimited list of field names. This View connects back to the Request-scoped Model, and retrieves one field value at a time:

```
<cfset data=request.data>
<tr>
<cfloop list="#attributes.fieldlist#"
  index="fieldname">
  <td>#data.getValue(fieldname)#</td>
</cfloop>
</tr>
```

A single nested View can now be used throughout the application whenever I want to output a table row with dynamic data. Changing the look and feel of the entire application can be accomplished with modifications to a single file.

Passing the Model Object

There isn't a single standard for how the Model object is passed to the View. In J2EE, where JSP pages are usually used as the View, it's common to place the Model in the Request scope, as it isn't possible to pass arguments to JSP files. In ColdFusion MX you can use custom tags as Views, so it would be valid (and perhaps preferable) to pass the CFC instance to the View as a custom tag attribute.

Each approach has its advantages. If the CFC instance is Request-scoped, each nested View isn't dependent on getting its data from its parent View, so you achieve more independence. But what if each View is supposed to get a different version of the Model? In this case it's probably better to pass the specific Model instance as an attribute, achieving better control.

More About Models

Macromedia recommends the use of CFCs in CFMX whenever you need functionality that's strictly server-side. Database queries, file management functions, communication with LDAP servers, and parsing of XML files are all examples of functions that manage data, not presentation. A CFC is ideally suited for use as a Model.

In the MVC pattern the Model is the most visible to, and the least aware of, the other members of the pattern. By "most visible" I mean that it can be seen and used by a Controller or a View. By "least aware" I mean that it doesn't need to know who is calling it.

In some MVC implementations the Model is able to notify Views of changes in state. In a ColdFusion implementation the View is short-lived (it doesn't stay in a persistent scope as the Model does, or as a View would in a persistent visual application). So in ColdFusion it only makes sense for the View to make requests of the Model and not the other way around.

Imagine that you've designed your MVC application from the beginning assuming that your user data is in a database table. Now you're told that the data will be moved to an LDAP (Lightweight Directory Access Protocol) server. Knowledge of the source and structure of your user information is programmed solely in the CFC and is hidden from the rest of your application. In object-oriented terms the data has been "encapsulated."

All required changes to your application will be localized to the CFC. As long as the syntax of the CFC method remains stable, the other members of the pattern (the Controllers and Views) don't have to be modified at all. Again, maintainability is significantly improved.

It's also possible to take advantage of CFC inheritance to create a clean, maintainable architecture. Notice that the file `user_db.cfc` (see Listing 2) includes only the `init()` method. All other methods are inherited from a "super" CFC (see Listing 3). The methods in this class are common to all of my data model CFCs. It would also be possible to create sibling CFCs for reading XML files or translating Web service calls. As long as they all deliver their data to the View with the same API, the Views don't need to change.

Caching and State

State is a big deal in Web applications. Whether you need to store user information for site personalization or improve performance with intelligent data caching, ColdFusion has always made it very easy to create and read persistent data using the Session and Application scopes.

Imagine that I want to cache the "users" table in server memory. Whenever a user record is added, updated, or deleted, the cached version should be refreshed from the database.

In a typical ColdFusion application that doesn't use components, the logic and variable names would be scattered among multiple ColdFusion pages. A CFC, however, can be created in the Application or Session scope, and can store data and logic in its own properties and methods.

In my `Application.cfm` page I might have this code:

```
<cfif not structkeyexists(application, "userdata")>
  <cfset application.userinfo=createobject("component",
    "user")>
</cfif>
```

As before, in the constructor code I could initialize a variable named "query" as a member of the component instance using the "this" prefix:

```
<cfset this.query="">
```

In the function that creates the query object of all users, I would first check to see whether the variable is a true query object. If it isn't a variable of type "query," I execute `<cfquery>` to refresh the data:

```
<cffunction name="init()">
  <cfif not isquery(this.query)>
    <cfquery name="this.query">
      SELECT * from users
    </cfquery>
  </cfif>
</cffunction>
```

If the CFC instance has been cached in the Application scope, the data is now available to all users of the Application. The first use of the `init()` method by any user will cause the query to be created. Subsequent users will get the query from memory.

The cached data can be cleared within the CFC whenever necessary. For instance, this CFC function is used to update user records:

```
<cffunction name="updateUser">
  <cfargument name="id" type="numeric">
  <cfargument name="name" type="string">
  <cfquery datasource="myDSN">
    UPDATE users
    SET user_name='#arguments.name#'
    WHERE user_id=#arguments.id#
  </cfquery>
  <cfset this.query="">
</cffunction>
```

PACIFIC ONLINE
www.paconline.net

Resetting the variable “this.query” to a string forces the query to be refreshed the next time the data is requested. This logic would be repeated whenever the data in the database table has been changed.

Caching logic is now restricted to the CFC. External users of the component simply ask for the data. It's up to the Model to determine whether the data should be read from database or memory. No matter how many times the data is called from various parts of the application, if the business logic or data storage mechanism changes, the programming changes will again be localized to the CFC and won't have any effect on the Views that make use of the data.

Conclusion

Every software design choice has both benefits and consequences. Adoption of

MVC-style programming makes your site more maintainable, but takes more initial time and thought than simply throwing a few ColdFusion pages together.

Also, any new design pattern should be load-tested early in the development process to ensure that the extra layers of code don't end up crippling the application's performance. But the first time you make a significant change to your application and find you can do it in just a few minutes without fear of collapsing the whole house of cards, the exercise pays off huge dividends.

In this article I've described some of my own experiments with the new CFMX toolset. As I mentioned at the beginning, I'm just beginning to explore the power and flexibility of the CFC programming model. Try it out and let me know what you learn. 🍷

Resources

- *ColdFusion components*: www.macromedia.com/desdev/mx/coldfusion/articles/intro_cfcs.html
- Alur, D., Crupi, J., and Malks, D. (2001). *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall.
- *J2EE Design Patterns*: http://java.sun.com/blueprints/patterns/j2ee_patterns/model_view_controller/
- *Software design patterns*: Design Patterns, Addison Wesley, 1995; www.macromedia.com/desdev/articles/design_patterns.html

About the Author

David Gassner is a Macromedia-certified instructor and developer in Napa, CA. He teaches classes in ColdFusion, HTML, Java, and XML as well as other Web development skills for Macromedia and corporate clients.

@ DGASSNER@NAPANET.NET

Listing 1

```
<!-- user_list.cfm
A Controller Page -->

<cfset request.data=createobject("component", "user_db")>
<cfif isdefined("url.sort")>
  <cfset request.data.setSortOrder(url.sort)>
</cfif>
<cfinclude template="user_list_view.cfm">
```

Listing 2

```
<!-- A Data Provider Component
This is a "child" CFC which inherits
most of its functionality from its
"super", dbmodel.

All variables with "this" prefix are
initialized in the super CFC.
-->

<cfcomponent extends="dbmodel" output="false">
  <cffunction name="init" output="false"
    access="public">
    <cftry>

      <!-- Query the database -->
      <cfquery name="this.query" datasource="#this.dsn#">
        SELECT * from users
      <cfif this.sortorder neq "">
        ORDER BY #this.sortorder#
      </cfif>
      </cfquery>

<cfcatch>
  <!-- If the query fails, create a dummy query -->
  <cfinclude template="user_dummyquery.cfm">
</cfcatch>

</cftry>
</cffunction>
</cfcomponent>
```

Listing 3

```
<!-- A "super" CFC, which is extended
      by CFCs bound to particular DB tables -->

<cfcomponent displayname="Data Model Super CFC"
      output="false">

<!-- Constructor code - initialize variables
      upon instantiation of the data object -->
<cfset this.currentrow=1>
<cfset this.query="">
<cfset this.sortorder="">
<cfset this.dsn="MyDSN">
```

```
<!-- Set the query sort order -->
<cffunction name="setSortOrder"
  output="false"  access="public">
  <cfargument name="sortorder" required="yes"
    type="string">
  <cfset this.sortorder=arguments.sortorder>
  <cfset this.query="">
</cffunction>
```

```
<!-- Advances "cursor" to next record if possible,
returns false if it's already there --->
<cfunction name="next" returntype="boolean"
access="public" output="false">
  <cfif not isquery(this.query)>
    <cfset this.init()>
    <cfset this.currentrow=1>
    <cfreturn true>
  <cfelseif this.currentrow eq
    this.query.recordcount>
    <cfreturn false>
  <cfelse>
    <cfset this.currentrow=this.currentrow+1>
    <cfreturn true>
  </cfif>
</cfunction>
```

```
<!-- Returns the value of a particular field
in the current row of the current table -->
<cffunction name="getValue" access="public"
output="false">
  <cfargument name="fieldname"
    required="true" type="string">
  <cfif not isquery(this.query)>
    <cfset this.init()>
  </cfif>
  <cfreturn
    this.query[arguments.fieldname][this.currentrow]>
</cffunction>
```

```
<!-- Return current row number -->
<cffunction name="getCurrentRow" returntype="numeric"
  access="public" output="false">
  <cfreturn this.currentrow>
```

Listing 4

```
<!-- user_list.cfm
  A View of a user list
  Depends on a data object being present in
  the request scope
-->
```

```
<cfset data=request.data>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html><head><title>User Information</title></head>
<body>

<table border="1">
<tr bgcolor="#FF66FF">
  <!-- In MVC terminology, these hyperlinks are
        messages sent from the View to the Controller -->
<cfoutput>
<th>
  <a href="#cgi.script_name#?sort=user_name">Name</a>
</th>
<th>
  <a href="#cgi.script_name#?sort=user_city">City</a>
</th>
</cfoutput>
</tr>
<cfloop condition="data.next()">
  <!-- A Nested View -->
<cf_view_row fieldlist="user_name,user_city">
</cfloop>
</table>

</body></html>
```

Listing 5

```
<!-- view_row.cfm Custom Tag
required attribute: fieldlist
(commma-delimited list of fields)
-->
<cfset data=request.data>
<cfif (data.getCurrentRow() mod 2)>
    <cfset bgcolor="white">
<cfelse>
    <cfset bgcolor="silver">
</cfif>
<cfoutput>
```

```
<tr bgcolor="#bgcolor#">
<cfloop list="#attributes.fieldlist#"
index="fieldname">
<td>#data.getValue(fieldname)#</td>
</cfloop>
</tr>
</cfoutput>
```

Listing 6

```
<!--- user_dummyquery.cfm
Creates a query object for testing
when the database isn't available
-->

<cfscript>
this.query=QueryNew("user_name, user_city");
queryAddRow(this.query);
querySetCell(this.query, "user_name", "David");
querySetCell(this.query, "user_city", "Napa");
queryAddRow(this.query);
querySetCell(this.query, "user_name", "Mary");
querySetCell(this.query, "user_city", "San Francisco");
queryAddRow(this.query);
querySetCell(this.query, "user_name", "Alex");
querySetCell(this.query, "user_city", "San Jose");
queryAddRow(this.query);
querySetCell(this.query, "user_name", "Lisa");
querySetCell(this.query, "user_city", "Los Angeles");
</cfscript>
```

```
<cfif this.sortorder neq "">
  <cfquery name="this.query" dbtype="query">
    SELECT * from this.query ORDER BY #this.sortorder#
  </cfquery>
</cfif>
```

CODE
LISTING

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

Pick 3 or More & SAVE BIG!

Web Services Journal • Java Developer's Journal
.NET Developer's Journal • XML-Journal
WebLogic Developer's Journal • WebSphere Developer's Journal
ColdFusion Developer's Journal • Wireless Business & Technology
PowerBuilder Developer's Journal

SPECIAL LOW PRICE
when you subscribe to 3
or more of our magazines

RECEIVE YOUR
DIGITAL EDITION
ACCESS CODE
INSTANTLY

WITH YOUR PAID
SUBSCRIPTION

www.sys-con.com/suboffer.cfm

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

ColdFusion MX by Macromedia

The Web's new fax machine?

REVIEWED BY
CAREY LILLY



CF has been rewritten using Java technology, and is much more than just a way for Web pages to interact with databases.

You could say that the fax machine revolutionized the business world. No more waiting for contracts as they made their slow progression from one mailbox to another. No more days and weeks passing as you shuttled layouts to your client, and they sent back the revisions and you sent back new layouts....It ate into companies' profits.

But with the fax machine, anything that could be put on a piece of paper could be sent anywhere in the world in a fraction of the time needed to physically ship it. There was one catch, of course: you needed a fax machine. Two, actually. For the fax machine to be the effective tool it became, there needed to be a fax machine in every office.

Now I have to tell you, I fell in love with ColdFusion back with version 1.0. I had come out of the dBase/Clipper environment, and CF was like an old friend. My very first CF application was a sort of shopping cart sometime before online shopping really caught on. Writing it was a cinch, considering that my other option at the time was Perl, which I was still largely unfamiliar with. The only catch to providing applications in CF is, of course, that you need a host that provides it. In the very beginning, CF didn't look to be poised to be the next fax machine.

That could be changing. With 75 of the Fortune 100 companies using ColdFusion, there is certainly a chance that CF will overtake more commonly available scripting languages like CGI or ASP. It seems likely to this developer, at least. Maybe I'm biased.

Since Macromedia acquired Allaire, I'm sure that the CF world has been awaiting with much anticipation (and perhaps apprehension?) the newest version of ColdFusion. ColdFusion MX, now a Macromedia product, has been released with greater connectivity to Web standards and powerful features to enhance the end user's experience. Integrated tools are available to put it all together. This product is more than repackaging or slapping on a new nametag.

This review is not a tag-by-tag description. I'm a developer who uses CF day in and day out. You other developers, you can listen up, even though I know you're probably already itching to get your hands on this. But hosting providers should pay attention as well. If you told your clients you didn't have a fax machine, you'd be subjected to a few seconds of stunned silence. Don't let that happen when people ask you if you provide CF.

Installing ColdFusion MX

The version I tested was CFMX beta 3. I installed it on a PC running Windows 2000. CFMX comes with a standalone Web server, which I used for my testing. The standalone server won't do the job as a functioning server, but for a development workstation it's more than adequate.

Since I already had CF 4.5 running on this system, the CFMX installer needed to migrate my settings. This was smooth, but it wasn't very fast as I have perhaps a dozen datasources set up. This may be a consideration for those migrating a server to the new version. Installation and migra-

tion were troublefree – much less painful than I had anticipated. Just so you know, you can test your code using the Code Compatibility Analyzer before you move to CFMX so you can identify any potential problems before you upgrade.

The first thing I did, of course, was run one of my existing CF applications. When I opened my eyes, why, there was my good old CF application, right where I left it and sitting quite happily on its new, more robust perch! What about the Java? Some people may tend to shy away from Java-based programs, since in past experience they're "too slow," or, in the words of one user, "they sucked." But even on my somewhat pedestrian workstation, CFMX provided speed equal to or better than my old version. This wasn't an official speed test, mind you, but it counted in my book. Additionally, I've seen other comments indicative of a performance increase using CFMX.

Some Cool Stuff

CFMX is "rearchitected" to run on the Java platform. This is certainly a Cool Thing, because it can not only encourage more developers who are looking for a wider audience, but it may also motivate more hosts to provide CF. My experience has been that, while CF has the power to provide what my clients want, finding a CF host can be a...well, let's just say that's it's been instructive. Granted, it's much easier to find a CF host than it was a few years ago, but if more hosts will come on board with CF, it may just become the Web's new "fax machine."

There are a number of new tags and functions in CFMX, including a whole set for XML support. First off, though, you'll want to wrap your brain around ColdFusion components (CFCs), which I thought were pretty cool. Now all of us, hopefully, develop CF apps using some sort of components (using CFINCLUDE or passing params using CFMODULE) to reuse common code. The problem comes when someone else wants to use our code. Some poor sap has to figure out what parameters to pass to your template, and what he'll get back (even though it's fully documented...right?). Okay, so maybe you don't document your components as well as you could. CFCs encapsulate their functionality so that you need only to call the component to display documentation about its methods and the parameters those methods require. If you're a developer...c'mon, self-documenting code? You'd be crazy not to want it.

Another thing to look at is Web services. CFMX provides easy-to-use syntax to hook up with Web services based on the Simple Object

VITALS

ColdFusion MX Server
Macromedia, Inc.

Address: 600 Townsend Street
San Francisco, CA 94103

Phone: 415 252-2000

Fax: 415 626-0554

Web: www.macromedia.com

Test Environment: Windows 2000 Server,
ColdFusion 4.5

Pricing: Professional Edition: \$799/server
Enterprise Edition: \$4,999/server
CFMX for J2EE: \$3,399/CPU
Upgrade from CF4.5 or higher at \$549
for Pro and \$2,499 for Enterprise

Access Protocol (SOAP). Using this standard, CFMX applications can interact with services using such technologies as ASP.NET, Java, or PHP. This is a Cool Thing for intranet developers who need to combine

various technologies. ColdFusion-based services can also be provided to these other platforms through the SOAP standard.

Before printing technology caught up, fax machines used that flimsy thermal paper. So not only did you need a new machine in every office, but now they all needed to have special media to display what you had. With CFMX you get native integration to the Flash player, which – according to Macromedia – is installed in 98% of the browsers accessing the Web. Sounds like the paper comes with the fax machine, eh?

Now, I have to say I'm not really a "rich content" sort of developer. For the most part, my clients' needs are simple, and simplicity is a way to keep costs and development time down. That said, I'm starting to think that developing rich client interfaces using Flash may be my new direction. Honestly, as the broadband market expands, users will expect a richer experience as they interact with your Web site. Web users are increasingly less technical and more consumer oriented. If you've got an

ABOUT THE AUTHOR

Carey Lilly is an associate with a Web site development firm based in the New York area. He has been developing with ColdFusion since 1997 and has 10 years' experience with relational databases.

SHOP ONLINE AT **JDJSTORE.COM** FOR BEST PRICES OR CALL IN YOUR ORDER AT **1 888-303-JAVA**

**BUY THOUSANDS OF
PRODUCTS AT
GUARANTEED
LOWEST PRICES!**

GUARANTEED BEST PRICES

FOR ALL YOUR SOFTWARE AND WIRELESS NEEDS



MACROMEDIA®

ColdFusion MX Server Pro
Multiplatform Upgrade

Rapidly build and deploy dynamic sites and rich Internet applications with ColdFusion MX. CFMX is a powerful environment for rapidly building and deploying dynamic sites and rich Internet applications. Its easy server scripting environment includes tag-based language, new server-side ActionScript, and complete integration with Dreamweaver MX (sold separately).



Macromedia® ColdFusion MX Server Pro**\$514****

MACROMEDIA®

JRun 4 Win/Linux/Unix
Full 1 CPU

Macromedia JRun 4 is the fast, affordable, and reliable solution for delivering Java applications. New features such as drag-and-drop deploy, XDoclet integration, and optimized Macromedia Flash connectivity speed the development and deployment of your next-generation Internet applications. Currently used in production at over 10,000 companies worldwide, JRun has confirmed reliability for powering J2EE applications.



Macromedia® JRun 4 Win/Linux/Unix.....**\$849****

MACROMEDIA®

ColdFusion Server 5 Enterprise
Multiplatform Upgrade

Rapidly build and deploy dynamic sites and rich Internet applications with ColdFusion MX. CFMX is a powerful environment for rapidly building and deploying dynamic sites and rich Internet applications. Its easy server scripting environment includes tag-based language, new server-side ActionScript, and complete integration with Dreamweaver MX (sold separately).



Macromedia® ColdFusion MX Server Enterprise**\$2319****

WWW.JDJSTORE.COM

Don't Miss the October issue of

COLD FUSION Developer's Journal

Error Handling

Debugging Flash/CFMX applications
HOW CAN YOU USE THE
NETCONNECT DEBUGGER?
Find out in *CFDJ* October

Error Prevention

Detect and prevent JavaScript errors
HOW IS IT DONE?
Find out in *CFDJ* October

<CFIF> Performance Measurements

Optimizing CFIF tasks
HOW DO THEY MEASURE UP?
Find out in *CFDJ* October

Product Review

Studio MX
WHAT'S NEW?
Find out in *CFDJ* October

online catalog, it had better be an easy, even fun, shopping experience or your customers may just go to your competition (or to the mall).

One difficulty I've had with online shopping apps is that everything has to be based on page calls, unless you're using browser-based scripting or something to augment your CF code. Using Flash integration, CF can interact with your application on the browser, with Flash as a gateway handling the client/server communication. Using Flash, you could write a drag-and-drop shopping application, making the experience more interactive and easier to use, and limiting the need for page reloads.

Naturally, making full use of all these great features requires the right tools. Dreamweaver MX, Flash MX, Fireworks MX, and Freehand 10 are available in a separate Studio MX package. A subsequent article will review these products.

So...

I used to think that since I already had a basic familiarity with languages similar to CF, my ease in learning it was because of that. Having been a lurker in CF newsgroups and on Macromedia's forums, I find that, along with seasoned coders, all types of programming newbies seem to gravitate to ColdFusion. It really is an easy language to learn. That right there is a good reason to use it.

I've been developing with CF for years. I have to admit that in the beginning it seemed a small chance that CF would last against "free" scripting languages like Perl and ASP. Each year, though, I've seen ".cfm" pages on more and more of the interactive sites that I visit. ColdFusion is not struggling against a current of ubiquitous and less expensive platforms. In fact, it's much more of an equal. Like the fax machine, general acceptance is necessary, but the future looks bright for CF.

I think every major release of ColdFusion has included something that wows me. Heck, I'm still impressed with "query of queries." I'm sure a number of developers are probably sitting on a stable system, happy with CF as it stands. A balance must be struck between the knee-jerk "ooh! A new toy!" reaction and the more sedate "let's see..."

“CFCs provide self-documenting taglike components, usable from any Web application”

reaction. That said, a trial version is available, and CFMX won't overwrite your old ColdFusion installation. My suggestion is to at least try it (you know you want to).

Perhaps you're on the fence about it. CFMX is not about ColdFusion, but about using CF along with all the other tools and services on the Web. There are sites that use Flash, and the owners of those sites may never have heard of ColdFusion. But CFMX can be used to make their sites better. CFMX adds functions to read and write XML, something every developer needs to do at one time or another. CFCs provide self-documenting taglike components, usable from any Web application. "Better results with less effort" is the Macromedia pitch. I've found it to be true for a few years now. CFMX opens up the door for lots of others to find that out as well.

For you intranet developers, especially those that are integrating multiple platforms, I think CFMX is a good move only because it expands your ability to interact with these platforms while providing the easy-to-learn CF scripting environment.

For you hosting companies, I'll give you a single compelling reason to buy or upgrade to CFMX: the fax machine. You could be one of the early providers of the "fax machine" of the Web, and people will come to your service because you provide what they want. Certainly, there are major corporations that use CF and they're another impetus. But what made fax machines work was not that IBM or Citibank had them. What made a difference is that in just a few years every office in the world had a fax machine, and your business was a curiosity if you didn't.

@ CAREY@WORLDCONTACT.COM

www.ColdFusionJournal.com

Web Services Java XML .NET Wireless

web services **EDGE**
conference & expo

JDJ **EDGE**
conference & expo

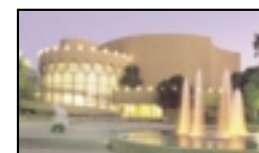
XML **EDGE**
conference & expo

wireless **EDGE**
conference & expo

The Largest Web Services, Java, XML, .NET and Wireless Conference and Expo

WWW.SYS-CON.COM
NEW LOWER REGISTRATION
THE BEST CONFERENCE BUY OF THE YEAR!

OWNED BY
SYS-CON MEDIA
PRODUCED BY
SYS-CON EVENTS



CONFERENCE: OCTOBER 1-3, 2002

EXPO: OCTOBER 2-3, 2002

SAN JOSE MCENERY CONVENTION CENTER, SAN JOSE, CA

Focus on Web Services

Those companies that get an early jump on Web services to integrate applications for the enterprise will be the winners in today's challenging landscape. Take three days to jump start your Web services education!

Focus on Java

Hear from the leading minds in Java how this essential technology offers robust solutions to i-technology professionals and senior IT strategy decision-makers.

Focus on XML

Here's your chance to hear up-to-the-minute developments in standards, interoperability, content management, and today's new quest for more efficient and cost-effective Internet and intranet-enabled business process integration.

Reaching Beyond the Enterprise

The Conference...

Organized into five comprehensive tracks:

Web Services, Java, XML, Wireless, and IT Strategy

- Over 60 Information-packed Sessions, Case Studies and tutorials
- Thought leaders and innovators presenting the topics you most want to hear

Free...

FREE Web Services Tutorial to the First 100 to Register.

Attend a FREE Web Services Tutorial on October 3.

Register by August 15th to reserve your seat!

Who Should Attend...

- Developers, Programmers, Engineers
- i-Technology Professionals
- Senior Business Management
- C Level Executives
- Analysts, Consultants

The Largest Expo...

Over 75 exhibitors will deliver their best product demonstrations, stand ready to hear your challenges, and help you find solutions. Don't miss this important opportunity to expand your options.

PLATINUM SPONSOR



GOLD SPONSORS



SILVER SPONSOR



CORPORATE SPONSORS



MEDIA SPONSORS



Tracking Traffic at Your Site

BY
DAVID
SHADOVITZ



Using a database and CF's graphing capability

As a member of Team Macromedia I respond to a lot of questions posted on the Macromedia Forums (<http://webforums.macromedia.com/coldfusion>).

My responses often contain sample code, and I realized that many questions arose over and over again. So rather than repeatedly typing the same code snippets, I created a small Web site (<http://members.evolt.org/dshadovi>) to hold them. Now I can simply point to the relevant page when I respond to a question.

Being a nosy guy, I wanted to know how the site was being used. Yes, there are tools that do a wonderful job of presenting this information, such as Analog and Web-Trends. But they require access to the Web server logs, which I don't have. Anyway, I wanted to roll my own. That's the fun of CF!

My first cut at tracking traffic was to use session management and CFMAIL. I set up Application.cfm to send me a message whenever a new session was started. It worked, but sometimes the mail was overwhelming. The biggest problem with this technique, though, was that it made it hard to perform analysis. I couldn't easily determine which page was getting the most hits, or how many visitors use Netscape.

The right technique, and the one I'll describe in detail, is to record the site traffic in a database and use CF's graphing capabilities to display it.

Recording the Site Traffic

Using MySQL, I created a single table to record information on each hit. Listing 1 shows the structure of that table. As you can see, each record contains the page visited, the visitor's IP address and browser (the user agent), and the date and time of the visit.

A record is inserted into that database table each time the site is hit. Listing 2 shows the code to do this.

This code belongs in Application.cfm.

Note that the INSERT statement doesn't include the time of the hit. When no value is specified for a field of type TIMESTAMP, which cannot be null, MySQL automatically inserts the current date and time. I could have passed the value #CreateODBC DateTime(Now())#, but it's more efficient to let the database take care of it.

You'll also see that I've massaged the name of the page a bit. For all of the pages on my site, CGLSCRIPT_NAME starts with "dshadovi/", so I strip this off. I also strip off the ".cfm" extension, since the name of the page, not the extension, is

what's important for the graphical display of the traffic.

Don't chide me for not using CFQUERYPARAM. I do use it, but I omitted it here in the interest of readability.

Displaying the Site Traffic

Now that the pieces are in place to record the site traffic, we're ready to write the code to graphically display it. Listing 3 is a complete CFML template that does exactly that. Figure 1 is a screenshot of this template in action; you can also see it live at <http://members.evolt.org/dshadovi/traffic.cfm>.

Exception Handling

An exception is an error that occurs at runtime. Most if not all programming languages offer a way to handle exceptions. The goal of this handling is to minimize the damage done by the exception. Sometimes it's possible to repair the problem or to work around it. If not, the application can at least provide a detailed error message.

ColdFusion provides CFTRY/CFCATCH, modeled on C's try/catch statements. Using this construct, the ColdFusion Server tries to execute a designated block of code, and if it catches any exceptions it executes another designated block of code.

I use CFTRY/CFCATCH in Listings 2 and 3. Let's take a closer look.

Listing 2 shows a snippet of code in Application.cfm, which inserts a record into the database. The record contains information about the current page visit. Now, the main goal of my Web site is to provide helpful code to other programmers. Tracking visits to the site is only a minor goal, mostly to satisfy my curiosity. I certainly wouldn't want problems with this minor goal to interfere with my major goal.

What kinds of problems might happen? Well, the database server could go down, or the database's transaction log may be full. Either of these would make it impossible to insert a record, but there's no

reason why that should render my site unusable. So I put the CFQUERY tag in a CFTRY block. If there are any runtime errors, the CFCATCH code block is executed. The CFCATCH block here is empty because there's no remedial action to take. If a record can't be inserted, that's okay. Execution continues on with whatever code follows the entire CFTRY block.

In Listing 2 the protected code block, a CFQUERY tag, did under-the-covers work. Not so for Listing 3, in which the protected code block is a CFGGRAPH tag. There is surely a user impact if the graph isn't displayed. So in this case the CFCATCH block isn't empty. Rather, it lets the user know there should have been a graph, and points to a possible cause of the problem. (Okay, I should also have code here to automatically e-mail the details of the problem to the server administrator, or to log the problem.)

I've shown here two uses of CFTRY/CFCATCH. Beyond this, CFTRY's TYPE attribute lets you take different actions based on the type of exception. The ColdFusion Server also provides CFCATCH variables that contain details of the exception. And CFTHROW gives you the power to declare that an exception has taken place.

Using CFTRY/CFCATCH to handle exceptions lets you compartmentalize your code, leading to more robust applications.

\$195
FOR SYS-CON
SUBSCRIBERS
BEST EDUCATIONAL VALUE
OF THE YEAR!

web services **EDGE**
world tour 2002

TO REGISTER: www.sys-con.com or Call 201 802-3069

Take Your Career to the Next Level!

EACH CITY WILL BE
SPONSORED BY A LEADING
WEB SERVICES COMPANY

Learn How to Create, Test and Deploy Enterprise-Class Web Services Applications

IF YOU
MISSED THESE...

BOSTON, MA (Boston Marriott Newton) **SOLD OUT!**
WASHINGTON, DC (Tysons Corner Marriott) **SOLD OUT!**
NEW YORK, NY (Doubletree Guest Suites) **SOLD OUT!**
SAN FRANCISCO, CA (Marriott San Francisco) **SOLD OUT!** **CLASSES ADDED**

BE SURE NOT TO
MISS THESE...

...COMING TO A CITY NEAR YOU

2002
LOS ANGELES **SEPTEMBER 19**
SAN JOSE AT WEBSERVICES EDGE 2002 .. **OCTOBER 1**
CHICAGO **OCTOBER 17**
ATLANTA **OCTOBER 29**
MINNEAPOLIS **NOVEMBER 7**
NEW YORK **NOVEMBER 18**
SAN FRANCISCO **DECEMBER 3**
BOSTON **DECEMBER 12**

2003
CHARLOTTE **JANUARY 7**
MIAMI **JANUARY 14**
DALLAS **FEBRUARY 4**
BALTIMORE **FEBRUARY 20**
BOSTON **MARCH 11**

REGISTER WITH A COLLEAGUE AND SAVE 15% OFF THE LOWEST REGISTRATION FEE.

TOPICS HAVE INCLUDED: Developing SOAP Web Services
Architecting J2EE Web Services

On May 13th, the San Francisco tutorial drew a record 601 registrations.

SHARPEN YOUR
PROFESSIONAL SKILLS.

KEEP UP WITH THE
TECHNOLOGY
EVOLUTION!

"Presented an excellent overview of Web services. Great inside knowledge of both the new and old protocols. Great insight into the code piece."

— Rodrigo Frontecilla

"Very articulate on the Web services SOAP topic and well-prepared for many questions. I've learned a lot from this seminar and I appreciate this seminar for my job. Thank you!"

— Kenneth Unpingco, Southern Wine & Spirits of America

"I liked the overview of Web services and the use of specific tools to display ways to distribute Web services. Good for getting up to speed on the concepts."

— B. Ashton, Stopjetlag.com

Echoed over and over by
Web Services Edge World Tour
Attendees:

"Good balance of theory and demonstration."

"Excellent scope and depth for my background at this time. Use of examples was good."

"It was tailored toward my needs as a novice to SOAP Web services – and they explained everything."

WHO SHOULD ATTEND:

- Architects
- Developers
- Programmers
- IS/IT Managers
- C-Level Executives
- i-Technology Professionals

SPONSOR A CITY!

Position your company as
a leader in Web services

Call 201 802.3066
to discuss how



PRESENT YOUR COMPANY'S EXPERTS TO AN EAGER AUDIENCE
READY TO LEARN FROM YOU! ACT TODAY!

i-TECHNOLOGY
EDUCATION

REGISTRATION FOR EACH CITY CLOSING THREE BUSINESS DAYS BEFORE EACH TUTORIAL DATE. DON'T DELAY. SEATING IS LIMITED. NON-SUBSCRIBERS: REGISTER FOR \$245 AND RECEIVE THREE FREE ONE-YEAR SUBSCRIPTIONS TO WEB SERVICES JOURNAL, JAVA DEVELOPER'S JOURNAL, AND XML JOURNAL, PLUS YOUR CHOICE OF BEA WEBLOGIC DEVELOPER'S JOURNAL OR WEBSERVICES DEVELOPER'S JOURNAL, A \$345 VALUE!

TO REGISTER:
www.sys-con.com
or Call 201 802-3069

CFC Best Practices & Tips

Guidelines for working with CFCs

BY RAYMOND CAMDEN **B**ecause ColdFusion components are such a new feature, developers may not yet have a good idea of what constitutes best practices.

This article attempts to rectify that situation by enumerating a set of guidelines that should be used when designing and working with CFCs, one of the most interesting new features of ColdFusion MX. Bear in mind that because they're so new, even experienced developers are debating what makes sense as a "best practice." These types of articles are always a matter of contention. I encourage you to contact me or discuss the article on any of the public lists, such as this magazine's **CFDJ-List**. The best way we as developers can improve our own code is by discussing our methods with others. With that in mind, let's begin...

CFC Methods

Normally a CFC is composed of a large set of methods. Let's discuss a few things to remember when building methods.

Security

By default, all CFC methods have an access="public" setting. A method can have four different types of access. *Public* means that the method can be called from the local server or from instances of the CFC. The other settings are private, package, and remote. *Private* methods can be called only from other methods in the CFC itself. *Package* methods are those that can be called from other CFCs in the same package. *Remote*, the most open setting, can be called from any source, including remote servers and Flash files.

It's important that you remember the default setting of "public." While it's not as open as "remote," it may not be what you intend when designing your CFC. A simple way to handle this is to set all methods

in the CFC to "private." This means your methods will be as secure as possible. "Open" up your methods only on a case by case basis. If you want to be truly anal (and when it comes to security, you can't be too anal), follow this practice even when you're "really, really" sure the method will be used publicly. Wait until you actually code the CFM that will call the CFC's method. At that point you can change the method being called.

Attributes

The CFFUNCTION tag takes multiple attributes, many of which are optional. It's best to explicitly set most, or all, of them. At its basic level, CFFUNCTION needs a "name" attribute. However, each of the following attributes should be passed:

- **Output:** This defines whether or not the method directly outputs to the caller. This should always be set, and always set to false. I'll explain why later. Another reason to set this to false is that it removes the white space generated by the method call.
- **Access:** I already explained why you should explicitly set this, and what value should be used.
- **ReturnType:** This specifies the data type that will be returned from the method. Setting this helps in two ways: first, if by some accident you don't return the correct type, it will throw an error. This will cut down on potential bugs. Second, it will be displayed when viewing the descriptor, which helps other developers who use your CFC.
- **Roles:** The roles attribute ties in to ColdFusion MX's roles-based security system. This is probably the only attribute that doesn't

really need setting. If your application is using roles-based security, you may want to consider setting this attribute to your highest "level" role. Again, the thinking is to make the method as secure as possible and open only what you must.

- **DisplayName:** This is another attribute that isn't quite necessary. If for some reason you want to give a more descriptive name to a method, you can use this attribute. It will be displayed only when you view the descriptor.
- **Hint:** Like DisplayName, this attribute can be used to help describe the method. It will be displayed with the descriptor, but is also very useful in the code itself.

Let's take a look at a method with the barest essentials set:

```
<cffunction name="cartTotal">
... method code ...
<cfreturn total>
</cffunction>
```

Now compare this to a method with most of the optional attributes set. It's much clearer what the method is doing and how it will return information.

```
<cffunction
name="cartTotal" output="false"
access="private"
returnType="numeric"
hint="This method calculates
the total price of all the
items in the shopping cart. It
simply loops over the cart and
multiplies item price by
quantity.">
... method code ...
<cfreturn total>
</cffunction>
```

We can also apply this same principle to the CFARGUMENT tag. CFARGUMENT, like CFFUNCTION, requires only the "name" attribute. However, you can – and should – pass the following optional arguments:

- **Type:** This defines the type for the argument. Like ReturnType in CFFUNCTION, it adds another level of protection to your code. It also makes things easier. If you specify a type of "array," for example, you have no need to check it yourself.
- **Required:** This defaults to false. I say specify it anyway. This way you're reminded that the arguments are not going to be required. This is the opposite of CFSRIPT-based UDFs, which state that any argument listed is automatically required.
- **Default:** Any optional argument should have some default value; otherwise, why even have the argument?
- **Hint:** Again, this will be helpful when viewing the descriptor or

reading the code. You may have an argument named "cartVector" that is just vague enough to drive you crazy a few weeks from now. Using hint will help you remember what in the heck that argument was meant to do.

Once again, let's take a look at a method that doesn't use these optional attributes:

```
<cffunction name="cartTotal">
<cfargument name="addSalesTax">
<cfargument name="state">
<cfargument name="discount">
... method code ...
<cfreturn total>
</cffunction>
```

Now consider the version in Listing 1. As you can see, the second version of the method is much clearer about what exactly it does.

Output vs Return

When calling a method on a CFC, you can display the output in multiple ways. The method itself

can CFOUTPUT data or it can return the data to the caller. You should never CFOUTPUT. Instead, data should be returned by using the CFRETURN tag. There are a number of reasons for this. If you ever encounter a case where you don't want the result of the method displayed immediately (maybe you want to store the result in a database), you'll have to reengineer your method and any calls to it (unless you use CFSAVECONTENT to suppress the output).

Another reason not to use CFOUTPUT is because of a bug with persistent components. If you store an instance of a component in a persistent scope (like the Session scope), and if you call a method that outputs instead of returns data, you'll only be able to call it once. Any subsequent call will not display the output.

The most important reason not to output directly from the CFC is Flash. Any data that isn't returned from the method will not be usable from Flash applications that use

CUT OUT AND
COMPLETE FORM ON BACK

FREE EXPO PASS!

\$75 VALUE

web
services
conference & expo

JDJ
conference & expo

XML
conference & expo

wireless
conference & expo

OCTOBER 1-3, 2002
SAN JOSE MCENERY
CONVENTION CENTER,
SAN JOSE, CA

Look Under the Hood and Beyond the Hype...

DIVE INTO WHAT'S HEADED YOUR WAY IN
JAVA, XML, .NET, WEB SERVICES AND WIRELESS

Who Should Attend:

- Developers
- Programmers
- Architects
- Engineers
- C Level Executives

EXPO HOURS:

OCT. 1.....11:00 AM – 6:00 PM

OCT. 2.....11:00 AM – 6:00 PM

OCT. 3.....11:00 AM – 4:00 PM

Conference & Expo Features

- The Most Significant Web Services and Wireless Technology Event of the Year!
- Sun Java™ Fast Path and Sun Java University™ Classes
- Two Concurrent Conference Programs for just one Registration Fee
- The largest Web Services, Java, XML, .NET and Wireless Expo in the World!

Platinum Sponsor:

Gold Sponsors:

Silver Sponsor:

Corporate Sponsor:

Educational Sponsor:

SYSCON EVENTS MEDIA

REGISTER ONLINE: WWW.SYS-CON.COM WITH CODE 8102

your CFC. By using CFRETURN, you guarantee that it will work with both your CFML files and your Flash files.

Use the Var Scope

If you've read anything about user-defined functions, you've probably seen mention of the "Var" scope. This is a scope created specifically for the lifetime of the function. By creating data in this scope, you help ensure that a UDF doesn't accidentally overwrite variables in the template. The Var scope is just as important when working with CFC methods. Consider the following CFC method:

```
<cffunction name="returnFoo"
output="false" access="public"
returnType="numeric"
hint="Returns the value of
Foo">
<cfset x = 1>
<cfset y = returnGoo(x)>
<cfreturn x>
</cffunction>
```

This function simply sets *x* to 1 and *y* to the value of another

method in the CFC, and then returns the value of *x*. You'd expect this method to return 1. However, guess what happens if returnGoo looks like the following:

```
<cffunction name="returnGoo"
output="false" access="public"
returnType="numeric"
hint="Returns the value of
Goo">
<cfset x = 100>
<cfreturn x>
</cffunction>
```

You might expect this to have no impact on our original function, returnFoo, but because the *x* value was never Var scoped, instead of getting 1 back, we get 100. To correct this we can simply add the Var qualifier to the sets in each method:

```
<cffunction name="returnFoo"
output="false" access="public"
returnType="numeric"
hint="Returns the value of
Foo">
```

```
<cfset var x = 1>
<cfset var y = returnGoo(x)>
<cfreturn x>
</cffunction>
```

```
<cffunction name="returnGoo"
output="false" access="public"
returnType="numeric"
hint="Returns the value of
Goo">
<cfset var x = 100>
<cfreturn x>
</cffunction>
```

Once this is done, calling returnFoo returns the value we expect, 1.

Don't forget that any and all Var statements must be made before any real code. They should be placed immediately after any CFARGUMENT tag.

Component Data

Another aspect of CFCs is the use of component data. This is data that persists for the lifetime of the CFC and is accessible to both the methods of the CFC and, potentially, the template using the CFC.

The This Scope

Two scopes can be used with CFCs. The first one, which will be used most of the time, is the This scope. Values can be defined in this scope just as in any other:

```
<cfset this.name = "Raymond">
```

Any variable defined in this scope is available to the calling template. For example:

```
<cfset theOb =
createObject("component","test")
>
<cfoutput>theOb.name</cfoutput>
```

Conversely, if you pass a value to a CFC, it will automatically be placed in the This scope. Consider:

```
<cfset theOb =
createObject("component","test")
>
<cfset theOb.foo = 1>
```

Once the CFSET command is run, any method will have access to foo in the This scope.

What are some things to consider when storing information in a CFC? Any information placed in the This scope is public. It can be read by the calling template and even changed. This is fine if you don't mind the information being manipulated. However, you may not always want that. If you have information that you want to persist in the CFC without possibly exposing

it, simply leave off the This scope. Imagine the following code in a CFC:

```
<cfset this.name = "Raymond">
<cfset id = createUUID()>
```

This code creates two variables in the CFC. The first is a public variable called name. The second variable, id, is not public. There isn't a real name for this scope. I refer to it as a private scope. It's not the same as the Variables scope and you can't CFDUMP it, but it's a useful way to store data and keep it separate from the public variables.

Using CFPROPERTY

The CFPROPERTY tag, technically, serves little purpose for CFCs. Its main use is to help define return values for Web services. However, it can serve a useful purpose for CFCs as well. The first thing to remember is that the CFPROPERTY tag will not actually do anything at runtime. Consider the following line of code:

```
<cfproperty name="numberOfLegs"
type="numeric" required="true">
```

While it looks like it may add a level of validation to the tag, it really just defines metadata for the CFC. In other words, it helps describe it. The only required attribute is name. Everything else simply helps describe the property. This can be useful in multiple ways. First of all, it shows up in the

descriptor. Second, every attribute passed in is available via the getMetaData function. This means you could write your own validation routines. Consider the following CFPROPERTY tag:

```
<cfproperty name="numberOfLegs"
type="numeric" required="true"
range="1,8">
```

The range attribute isn't a real attribute, but it will show up in the metadata. It would be trivial then to write a method that validates the value for this.numberOfLegs to ensure that it falls between 1 and 8. As you can imagine, this is pretty open ended. Any attribute to CFPROPERTY can be passed and used. How it's implemented is completely up to the developer.

That's Not All, Folks...

As I said at the beginning, CFCs are a new feature. What makes sense to me, and others, at this point will seem a bit silly next year (or even a few months from now). If you have any ideas you'd like to add, or perhaps you see something you disagree with, please e-mail me. You should also visit www.CFCZone.org. This is a site run by Rob Brooks-Bilson. (It's in the same vein as www.CFLib.org.) The site will soon host free, and open source, CFCs that you can use in your own projects.

@JEDIMASTER@MACROMEDIA.COM

ABOUT THE AUTHOR

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

TO REGISTER:

ONLINE

WWW.SYS-CON.COM

FAX

201-782-9651

MAIL

SYS-CON EVENTS, INC.
135 CHESTNUT RIDGE ROAD
MONTVALE, NJ 07645

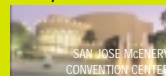
CALL

201-802-3069



OCTOBER 1-3, 2002

SAN JOSE MCENERY
CONVENTION CENTER,
SAN JOSE, CA



YOUR INFORMATION (Please Print)

☐ Mr. ☐ Ms.

First Name _____ Last Name _____
Title _____ Company _____
Street _____
Mail Stop _____
City _____
State _____ Zip _____ Country _____
Phone _____ Fax _____
E-mail _____

A. Your Job Title

- ☐ CTO, CIO, VP, Chief Architect
- ☐ Software Development Director/Manager/Evangelist
- ☐ IT Director/Manager
- ☐ Project Manager/Project Leader/Group Leader
- ☐ Software Architect/Systems Analyst
- ☐ Application Programmer/Evangelist
- ☐ Database Administrator/Programmer
- ☐ Software Developer/Systems Integrator/Consultant
- ☐ Web Programmers
- ☐ CEO/COO/President/Chairman/Owner/Partner
- ☐ VP/Director/Manager Marketing, Sales
- ☐ VP/Director/Manager of Product Development
- ☐ General Division Manager/Department Manager
- ☐ Other (please specify)

B. Business/Industry

- ☐ Computer Software
- ☐ Computer Hardware and Electronics
- ☐ Computer Networking & Telecommunications
- ☐ Internet/Web/E-commerce
- ☐ Consulting & Systems Integrator
- ☐ Financial Services
- ☐ Manufacturing
- ☐ Wholesale/Retail/Distribution
- ☐ Transportation
- ☐ Travel/Hospitality
- ☐ Government/Military/Aerospace
- ☐ Health Care/Medical
- ☐ Insurance/Legal
- ☐ Education
- ☐ Utilities
- ☐ Architecture/Construction/Real Estate
- ☐ Agriculture
- ☐ Nonprofit/Religious
- ☐ Other (please specify)

C. Total Number of Employees at Your Location and Entire Organization (check all that apply):

Location	Company
10,000 or more	<input type="checkbox"/> 01 <input type="checkbox"/> 01
5,000 - 9,999	<input type="checkbox"/> 02 <input type="checkbox"/> 02
1,000 - 4,999	<input type="checkbox"/> 03 <input type="checkbox"/> 03
500 - 999	<input type="checkbox"/> 04 <input type="checkbox"/> 04
100 - 499	<input type="checkbox"/> 05 <input type="checkbox"/> 05
100 or less	<input type="checkbox"/> 06 <input type="checkbox"/> 06

D. Please indicate the value of computer and communications products and services that you recommend, buy, specify or approve over the course of one year:

- ☐ \$10 million or more
- ☐ \$1 million - \$9.9 million
- ☐ \$500,000 - \$999,999
- ☐ \$100,000 - \$499,999
- ☐ \$10,000 - \$99,999
- ☐ Less than \$10,000
- ☐ Don't know

E. What is your company's gross annual revenue?

- ☐ \$10 billion or more
- ☐ \$1 billion - \$9.9 billion
- ☐ \$100 million - \$999 million
- ☐ \$10 million - \$99.9 million
- ☐ \$1 million - \$9.9 million
- ☐ Less than \$1 million
- ☐ Don't know

F. Do you recommend, specify, evaluate, approve or purchase wireless products or services for your organization?

- 01 ☐ Yes 02 ☐ No

\$75 Value

Complete and return this form to receive FREE admission to the Expo and all FREE Show Features.

After September 20, 2002, bring this pass onsite for FREE admission.

PLEASE PRINT CLEARLY
(PHOTOCOPY FOR ADDITIONAL REGISTRANTS)

G. Which of the following products, services, and/or technologies do you currently approve, specify or recommend the purchase of?

- ☐ Application Servers
- ☐ Web Servers
- ☐ Server Side Hardware
- ☐ Client Side Hardware
- ☐ Wireless Device Hardware
- ☐ Databases
- ☐ Java IDEs
- ☐ Class Libraries
- ☐ Software Testing Tools
- ☐ Web Testing Tools
- ☐ Modeling Tools
- ☐ Team Development Tools
- ☐ Installation Tools
- ☐ Frameworks
- ☐ Database Access Tools/JDBC Devices
- ☐ Application Integration Tools
- ☐ Enterprise Development Tool Suites
- ☐ Messaging Tools
- ☐ Reporting Tools
- ☐ Debugging Tools
- ☐ Virtual Machines
- ☐ Wireless Development Tools
- ☐ XML Tools
- ☐ Web Services Development Toolkits
- ☐ Professional Training Services
- ☐ Other (Please Specify)

SYS-CON EVENTS, INC. AND SYS-CON MEDIA MAKE NO GUARANTEES REGARDING CONTENT. SPEAKERS OR ATTENDANCE. THE OPINIONS OF SPEAKERS, EXHIBITORS AND SPONSORS DO NOT REFLECT THE OPINION OF SYS-CON EVENTS AND SYS-CON MEDIA AND NO ENDORSEMENT OF SPEAKERS, EXHIBITING COMPANIES' PRODUCTS, OR SPONSORS IS IMPLIED.

Listing 1

```
<cffunction name="cartTotal" output="false"
access="private" returnType="numeric" hint=
"This method calculates the total price of all
the items in the shopping cart. It simply loops
over the cart and multiplies item price by quanti-
ty."
>
... method code ...
<cfreturn total>
</cffunction>
```

```
<cffunction name="cartTotal" output="false"
access="private" returnType="numeric" hint=
"This method calculates the total price of all
the items in the shopping cart. It simply loops
over the cart and multiplies item price by quanti-
ty."
>
<cfargument name="addSalesTax" type="boolean"
default="false" required="false" hint=
"Should we add sales tax to the cart total.
```

If true, argument state will be required.">

```
<cfargument name="state" type="string"
default="" required="false" hint=
"State to use when applying sales tax. Will be
needed if addSaleTax=true.">
```

```
<cfargument name="discount" type="numeric"
default="0" required="false" hint=
"Numeric discount to apply to cart total.">
```

```
... method code ...
<cfreturn total>
</cffunction>
```

CODE LISTING

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

Ask the Training Staff

A source for your CF-related questions

BY
BRUCE
VAN HORN



September means summer is over and we all get back to work or back to school.

And for those of us who have been out of school for a very long time, I hope you always look for opportunities to learn something new. Send me your questions if you get stuck!

Q: I have a CF Scheduler question: How do I schedule the execution of a template to run at 6 a.m. on weekdays (Mon–Fri) only? I know how to schedule it at 6 a.m. every day, but I don't want it to run on Saturday or Sunday. Any suggestions?

A: Unfortunately, there's no way to get the CF Scheduler to do this for you (at least none that I'm aware of). There is, however, an easy way to accomplish this. Schedule the template to run at 6 a.m. every day, which you already know how to do, but put a CFIF block in your template to test for the day of the week. If the current day is a Saturday or Sunday, don't execute the code. The template is still executed by the scheduler, but none of the code executes on those days. Here's the code you need to wrap around the existing code in your page:

```
<CFIF DayOfWeek(Now()) gt 1 AND
DayOfWeek(Now()) lt 7>
...this will only run Mon–Fri!
</CFIF>
```

Q: I want to assign a user a password using a randomly generated string of letters and numbers without repeating any characters. How can I do this?

A: Great question. The solution is in the following code. Let's assume that the password will be eight characters long using both letters and

numbers. First, assign a variable that contains all the possible characters (I like to omit the letters "L" and "O" because they're often confused with the numbers 1 and 0). Next, initialize your password to null. Then loop from 1 to 8. For each iteration of the loop, pick a random number between 1 and the length of your character list. Select the character from the list based on the random number and append it to the password. Last, delete that character from the list so it won't get picked again.

```
<CFSET charList =
"a,b,c,d,e,f,g,h,i,j,k,m,n,p,q,r
,s,t,u,v,w,x,y,z,0,1,2,3,4,5,6,7
,8,9">
<CFSET Password = "">
<CFLOOP FROM="1" TO="8"
INDEX="i">
  <CFSET RandNum =
RandRange(1,ListLen(charList))>
  <CFSET Password = Password &
ListGetAt(charList,RandNum)>
  <CFSET charList =
ListDeleteAt(charList,RandNum)>
</CFLOOP>
<CFDUMP VAR="#Password#">
```

Q: We recently received an e-mail from one of our customers explaining that attaching the following code (mode=debug) to the URL of a ColdFusion file (for example, listings.cfm? mode=debug) will reveal most, if not all, of our network information (IP address, server information, OS information, etc). Obviously, this is a potential security threat and has shaken us. What can we do about it?

A: It's funny (ironic) that when this was first introduced (in version 4, I think), it was actually billed as a "feature"!



Appending "?mode=debug" to the URL does reveal the execution time of a page and all of the Form, URL, and CGI variables that were available to the page when it was processed. At times when debugging an application it's helpful to a developer to get this info on-the-fly by applying this URL string. But, as you mention, it can be very dangerous to your site if a good hacker gets this information! Fortunately, you can prevent this from happening by placing one line of code in your Application.cfm file:

```
<CFSETTING SHOWDEBUGOUTPUT="No">
```

With this tag in Application.cfm, nobody (including yourself) will be able to see any of this debug information. When developing an application, I leave this setting set to "Yes," but when I put the application into production, I set it to "No."

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

@BRUCE@NETSITEDYNAMICS.COM

www.ColdFusionJournal.com

introductory
subscription offer!

coming soon...

A TRULY INDEPENDENT VOICE IN THE WORLD OF .NET

.NET Developer's Journal is the leading independent monthly publication targeted at .NET developers, particularly advanced developers. It brings .NET developers everything they need to know in order to create great software.

Published monthly, *.NET Developer's Journal* covers everything of interest to developers working with Microsoft .NET technologies – all from a completely independent and nonbiased perspective.

Articles are carefully selected for their prime technical content – technical details aren't watered down with lots of needless opinion and commentary. Apart from the technical content, expert analysts and software industry commentators keep developers and their managers abreast of the business forces influencing .NET's rapid development.

Wholly independent of both Microsoft Corporation and the other main players now shaping the course of .NET and Web services, *.NET Developer's Journal* represents a constant, neutral, expert voice on the state of .NET today – the good, the bad, and the ugly...no exceptions.

Here's what you'll find in every issue of .netdj:

- Security Watch
- Mobile .NET
- .NET Trends
- Tech Tips
- Standards Watch
- Business Alerts
- .NET News
- Book and Software Announcements

.NET Developer's Journal is for .NET developers of all levels, especially those "in the trenches" creating .NET code on a daily basis:

- For beginners: Each issue contains step-by-step tutorials.
- For intermediate developers: There are more advanced articles.
- For advanced .NET developers: In-depth technical articles and columns written by acknowledged .NET experts.

Regardless of their experience level, *.NET Developer's Journal* assumes that everyone reading it shares a common desire to understand as much about .NET – and the business forces shaping it – as possible. Our aim is to help bring our reader-developers closer and closer to that goal with each and every new issue!

SUBSCRIBE ONLINE!

www.sys-con.com/dotnet/

or Call

1-888-303-5282

SAVE 16% OFF

THE ANNUAL COVER PRICE

Get 12 issues of **.NETDJ**
for only \$69⁹⁹!

ANNUAL
COVER PRICE:

~~\$83.88~~

YOU PAY

\$69⁹⁹

YOU SAVE

\$13.89

OFF THE ANNUAL
COVER PRICE

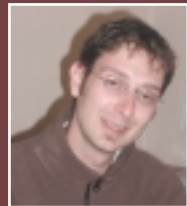
*OFFER SUBJECT TO CHANGE WITHOUT NOTICE

ABOUT THE
AUTHOR
Bruce Van Horn is president of Netsite Dynamics, LLC, a certified Macromedia developer/instructor, and a member of the CFDJ International Advisory Board.



CF Community

Tales from the List



BY SIMON HORWITH

Toward the end of July I posted the URL to and a summary of Macromedia's performance comparison between ColdFusion MX and ColdFusion 5. The two versions of ColdFusion were performance tested on identical machines, with several different processor and memory configurations, on the Windows, Solaris, and Linux operating systems. The application servers had trusted cache enabled (typical only of a production server), and simulated a hundred user sessions (each with heavier than average real-world use), browsing an e-commerce application product catalog, selecting item(s) to purchase, and "checking out" their order. The overall results showed that CFMX averaged 10% faster on Windows 2000, 30% faster on Solaris, and 45% faster on Linux! As the performance comparison pointed out, this is even more impressive when you take into consideration the fact that CF5 was roughly 500% faster than the prior version (CF4.5 running SP2) and that CFMX is a complete server rewrite with a new underlying architecture and significant added functionality. The added functionality was also tested. ColdFusion components were compared with user-defined functions and custom tags. The tests show that no performance is lost with the new functionality in CFMX. Also worth noting is that CFMX shows more linear SMP scalability. This means that performance increases more with CFMX as more processors are added to a server than with prior versions of ColdFusion. I recommend reading the performance comparison for yourself at www.macromedia.com/software/coldfusion/whitepapers/pdf/cfmix_performance_brief.pdf.

Eugen Notter, a frequent list contributor, responded with an e-mail describing performance problems with a specific application he wrote. Seems this application has more than doubled in execution time after he upgraded the server from ColdFusion 5 to CFMX. What Eugen has found is that templates that do a lot of cfincludes, especially if the included files also do a lot of cfincludes, are executing very slowly compared to their execution time on CF5. Morgan Kelsey replied, noting his

MX outshines the rest!

personal observation that after migrating a server from 5 to MX, applications he'd written with more efficient code performed very well, but applications he'd written years ago that use more nested tags and includes performed much worse on MX than on CF5. Kevin Bridges suggested that the overhead sometimes associated with object-oriented design as opposed to sequential programming might be a possible explanation for this server behavior. Jeffrey Houser, another very active and highly respected list member, quickly pointed out that the server was written in C++ before the MX version, and that unless the code base was procedural C (which is very unlikely), object-oriented design overhead shouldn't be blamed.

Nobody is 100% sure exactly why Eugen is seeing this behavior, but others on the list also responded, noting that they are experiencing similar performance issues with CFMX when cfincluding a lot of files, nesting cfincludes, and/or nesting custom tags. Unfortunately, the short answer for now is going to have to be "don't do that." The good news is that applications written specifically for MX should be leveraging the component architecture, which should result in far fewer (if any) nested tags or cifs.

...

Aside from this performance issue with nested included files and nested tags, everyone on the list has been extremely impressed with the performance of ColdFusion MX. The only other performance issues I've seen have been with COM objects (objects called through the MX Java-COM bridge are very slow) and when using the JDBC-to-ODBC bridge, which is another ill-advised practice. Use Java objects and JDBC drivers and these two issues are...nonissues. Now that there's an "official" performance comparison between CF5 and MX, we developers have confirmation of what many of us already suspected (and some claimed to know already)...the new MX server is not only finally here, but it kicks butt and is here to stay!

@ SHORWITH@FIGLEAF.COM

Subscribe Now and **SAVE!**

ONLY **\$89⁹⁹** for 12 monthly issues

Special Limited Time Offer

Here's what you'll find in every issue of

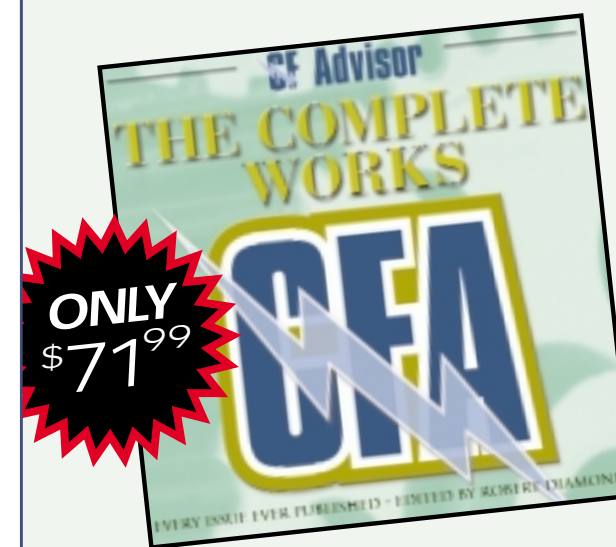
COLDFUSION
Developer's Journal:

- New Features of ColdFusion MX
- Tips, Techniques, and Tricks in Server-Side Programming
- DB Programming Techniques in CF
- High-Quality Technical Features
- <BF on CF> Column by Ben Forta
- Interviews with CF Makers and Shakers
- Display Ads of the Best Add-On Products
- CF User Group Information
- Product Reviews of the Best CF Add-Ons
- Q&A with ColdFusion Experts
- Full Working Source Code Samples

\$17⁸⁹ OFF
the Annual
Newsstand Rate



THE MOST COMPLETE LIBRARY OF EXCLUSIVE CF ADVISOR ARTICLES ON ONE CD!



ONLY
\$71⁹⁹

"The Complete Works"

CD is edited by *ColdFusion Developer's Journal*
Editor-in-Chief Robert Diamond
and organized into 21 chapters containing
more than 200 exclusive *CF Advisor* articles.

Easy-to-navigate HTML format!

- | | |
|---------------|------------------------|
| E-Commerce | Wireless |
| Interviews | Verity |
| Custom Tags | Source Code CF |
| Fusebox | Applications |
| Editorials | Object-Oriented CF |
| Databases | WDDX |
| News | Upgrading CF |
| CF & Java | Programming Tips |
| CFBasics | CF Tips & Techniques |
| Reviews | Programming Techniques |
| Scalability | Forms |
| Enterprise CF | |



135 CHESTNUT RIDGE ROAD
MONTVALE, NJ 07645
WWW.SYS-CON.COM

Order Online at
WWW.JDJSTORE.com
OFFER SUBJECT TO CHANGE WITHOUT NOTICE

*Now
on CD!*

**3
YEARS**

**40
ISSUES**

**200
ARTICLES**

**ONE
CD**



The Joy of Dreamweaver MX Published

(Berkeley, CA) – McGraw-Hill/Osborne Media has announced the publication of *The Joy of Dreamweaver MX: Recipes for Data-Driven Web Sites* by Paul Newman.

Written for the beginning and intermediate Dreamweaver MX user, the book offers step-by-step “recipes” for creating data-driven Web sites using ASP and ColdFusion MX, from defining a site’s goals to design and production to publishing and maintaining the active site.

The table of contents and a sample chapter are available at <http://shop.osborne.com/cgi-bin/osborne/0072224649.html>

www.mcgraw-hill.com

Macromedia’s Pet Market Application

(San Francisco) – Macromedia, Inc. introduces Pet Market, a sample application built using the entire Macromedia MX product family. Pet Market, a fictitious online



pet retailer, illustrates both the benefits of rich Internet applications and the ability to deliver these experiences across different server architectures, including the Java platform and Microsoft .NET Framework.

CFDJ ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
ACTIVE PDF	WWW.ACTIVEPDF.COM	949.582.9002	4
CF ADVISOR	WWW.CFADVISOR.COM	201.802.6069	49
CFDYNAMICS	WWW.CFDYNAMICS.COM	866.CFDYNAMICS	21
CFXHOSTING	WWW.CFXHOSTING.COM	866.CFX.HOST	27
E-ZONE MEDIA	WWW.FUSETALK.COM	866.477.7542	25
HOSTMYSITE.COM	WWW.HOSTMYSITE.COM	877.215.HOST	19
INTERLAND	WWW.INTERLAND.COM	800.845.8706	2
INTERMEDIA	WWW.INTERMEDIA.NET	800.379.7729	52
JDJ STORE	WWW.JDJSTORE.COM	201.802.3069	33, 35
MACROMEDIA	WWW.MACROMEDIA.COM/GO/CFMXAD	877.460.8679	9
MACROMEDIA	WWW.MACROMEDIA.COM/GO/CFDJDEVCON2002	877.460.8679	13
MACROMEDIA	WWW.MACROMEDIA.COM/GO/CFMXLIGHT	877.460.8679	23
.NETDJ	WWW.SYS-CON.COM/DOTNET/	888.303.5282	47
NEW ATLANTA	WWW.NEWATLANTA.COM		3
PACIFIC ONLINE	WWW.PACONLINE.NET	877.503.9870	31
PAPERTHIN	WWW.PAPERTHIN.COM	800.940.3087	15
SYS-CON MEDIA	WWW.SYS-CON.COM/SUBOFFER.CFM	201.802.3069	33
TERATECH INC.	http://WWW.CFCONF.ORG/CF_UNDERGROUND4/	800.447.9120	17
WEB SERVICES EDGE WEST	WWW.SYS-CON.COM	201.802.3069	37, 39, 43

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

DevCon 2002 Coming in October

(San Francisco) – Registration is now open for Macromedia Developer Conference 2002 (DevCon 2002), the premier event for Macromedia designers and developers to extend their skills and expand their knowledge base. The conference will feature more than a hundred sessions designed to provide hands-on training and skills development for developers and designers using Macromedia products. The conference will be held October 27–30 at the Walt Disney World Swan and Dolphin in Lake Buena Vista, Florida.

“With the launch of the Macromedia MX products this spring, there is a wide range of new skills and techniques that developers and designers will want to learn,” said Kevin Lynch, chief software architect, Macromedia. “DevCon 2002 is a unique opportunity to get intensive training on the technology, access to the information from leaders in the developer community, and first-hand information on the long-term Macromedia product strategy and technology vision.”

Register online at www.macromedia.com/go/devcon



Pet Market is accompanied online by source code and more than a dozen articles to show designers and developers how to create their own rich Internet applications that drive down cost and bandwidth concerns while providing a usable, high-impact user experience. The Pet Market site and resources can be viewed and downloaded from www.macromedia.com/desdev/mx/blueprint

Molecular Now a Macromedia Premier Partner

(Watertown, MA) – Molecular, an Internet services firm, has been selected by Macromedia as a Premier Partner in the Macromedia Alliance.

As part of the relationship, Molecular will utilize the Macromedia MX product family to develop appli-

cations that exceed the expectations of today’s online customer. Molecular will also partner with Macromedia on extensive client engagements with businesses in multiple vertical markets across the Northeast. www.molecular.com

EVENTS Macromedia DevCon 2002

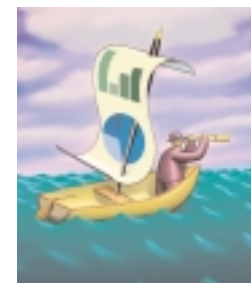
October 27–30
Lake Buena Vista, FL
www.macromedia.com/v1/conference/

CF Cruise 2003
Caribbean cruise –
departs from Tampa
www.cfconf.org/cf_cruise

Chart Your Course to Success in IT... ...order your copy of Java Trends: 2003



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



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

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

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

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

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

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

Navigate your company through the challenging IT marketplace with the trusted knowledge and intelligence of EnginData Research. Order your copy of the 2002–2003

Java market study by calling Margie Downs at 201-802-3082, or visiting our Web site.



www.engindata.com

INTERMEDIA.NET
www.intermedia.net