

COLDFUSION Developer's Journal

ColdFusionJournal.com

March 2007 Volume:9 Issue: 3

Object-Oriented Form Validation

22

Binary Data,
ColdFusion & Flex 7

Inheritance: Code
Reuse for CFCs 12

Building the Right Project
Team Is Key to a Successful
Technical Implementation 20

Don't Talk to Strangers! 28

Objects Everywhere 32



3-DAY EVENT!

SOA WORLD 2007
CONFERENCE & EXPO

ENTERPRISE 2007
OPENSOURCE
CONFERENCE & EXPO

2007 VIRTUALIZATION
CONFERENCE & EXPO
www.virtualizationconference.com

June 25-27, 2007
Roosevelt Hotel / New York City

SEE PAGE 17 & 29

Presorted
Standard
US Postage
PAID
St. Croix Press





Say hello to the next generation.

It's found in the world's most inspirational places. It's pushing the boundaries of art direction and design. Introducing Adobe® Creative Suite® 2.3. Today's most talented designers and art directors are working smarter and faster because of it. Just ask the masterminds behind INTERspectacular—they rely on the Creative Suite to help bring their ideas into the world. See how they do it at adobe.com/creativemind. It's everything but the idea. Better by Adobe.™

Luis Blanco and Michael Uman,
INTERspectacular

adobe.com/creativemind

©2006 Adobe Systems Incorporated. All rights reserved. Adobe, the Adobe logo, Creative Suite and Better by Adobe are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.



ColdFusion Hosting is our Complete Focus

➤ **POWERFUL HOSTING PLANS**

FREE SQL server access | FREE account setup | Unlimited email accounts Generous disk space & data transfer
30 day money-back guarantee | Great value

➤ **RELIABLE NETWORK**

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

➤ **FANTASTIC SUPPORT SERVICES**

24/7 support services | Knowledgeable phone support | We focus on your individual needs

CFDynamics

866.233.9626 ➤ CFDYNAMICS.COM

For years we have been involved in the Cold Fusion community and have come to know what developers and project managers look for in a web host. The combination of our powerful hosting plans, reliable network, and fantastic support sets us apart from other hosts.

Real service. Real satisfaction. Real value. Real support. Real Freedom.



editorial advisory board

Jeremy Allaire, *founder emeritus, macromedia, inc.*
Charlie Arehart, *CTO, new atlanta communications*
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**

Simon Horwith simon@sys-con.com

executive editor

Nancy Valentine nancy@sys-con.com

research editor

Bahadır Karuv, PhD bahadir@sys-con.com

production**lead designer**

Abraham Addo abraham@sys-con.com

art director

Alex Botero alex@sys-con.com

associate art directors

Louis F. Cuffari louis@sys-con.com

Tami Beatty tami@sys-con.com

editorial offices**SYS-CON MEDIA**

577 Chestnut Ridge Rd., Woodcliff Lake, NJ 07677
Telephone: 201 802-3000 Fax: 201 782-9638
COLD FUSION DEVELOPER'S JOURNAL (ISSN #1523-9101)
is published monthly (12 times a year)
by SYS-CON Publications, Inc.

postmaster: send address changes to:

COLD FUSION DEVELOPER'S JOURNAL
SYS-CON MEDIA
577 Chestnut Ridge Rd., Woodcliff Lake, NJ 07677

©copyright

Copyright © 2007 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.

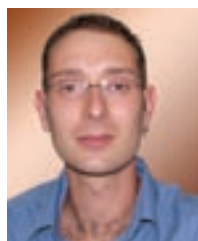
Worldwide Newsstand Distribution

Curtis Circulation Company, New Milford, NJ
FOR LIST RENTAL INFORMATION:
Kevin Collopy: 845 731-2684, kevin.collopy@edithroman.com
Frank Cipolla: 845 731-3832, frank.cipolla@epostdirect.com

For promotional reprints, contact reprint
coordinator Megan Mussa, megan@sys-con.com.
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.

Career Growth 101 for the ColdFusion Developer



By Simon Horwith

A student recently asked me what he could do to further his career, and inquired as to how I got to the point

I am at on my own. This is a subject generally reserved for informal chit-chat among developers at conference social events and after user group meetings, and is one that is very important to each of us.

It also brings up the fundamental question of whether or not there is a proven, easy-to-follow formula for career success as a CF developer.

The truth is that there is not any set formula to achieving success, as we each define success differently. Success for some is measured solely by their financial compensation and status. For others, success means happiness on the job and with their life – perhaps they place more emphasis on the people and environment they work in every day as well as on their vacation and other “personal luxury” benefits. Still others describe success purely by notoriety and demand – on both their achievements and their reputation.

For most of us, our career objective is in reality a combination of all or some of these goals. Obviously, everyone values their paycheck and most of us are hoping to see our salaries increase over time. Of course, money does you no good without a personal life and liberties to enjoy that hard-earned money outside the workplace, so some emphasis must be placed on quality of life. For many

developers their job is just their job, and they don't take any interest in public reputation or pride in their achievements, so long as the money and quality of life is satisfactory. Those who do measure success this way most often do so knowing that a good reputation and great achievements shown on paper also translate to better pay and negotiating power when discussing compensation with employers and prospective employers.

Though the goals and path necessary to achieve these goals may differ from developer to developer, I do have some sound advice and career exercises that will aid you whatever your goals may be.

Financial goals are best determined by evaluating where you are now, and determining where you want to be in, say, five years. If what you want is to retire in five years, then you need to do one of two things. You can work for a company you believe is going to go public, get bought, or do so well publicly that the pay-off will be enough to retire on. You also need to assure yourself that you have an employment contract with a company like this that guarantees you enough interest in the company that your interest will be worth enough to retire on. If you are fortunate enough to find a company and position like this, you are on your way. The other approach is to develop a commercial product yourself in the form of off-the-shelf software or a Web site that will generate revenue through advertising, membership/services fees, and/or being bought.

— CONTINUED ON PAGE 10

About the Author

Simon Horwith is the editor-in-chief of ColdFusion Developer's Journal and is the CIO at AboutWeb, LLC, a Washington, DC based company specializing in staff augmentation, consulting, and training. Simon is a Macromedia Certified Master Instructor and is a member of Team Macromedia.
simon@horwith.com

Your Web site ...



Your Web site Powered by Hot Banana



Hot Banana Web Sites Are Built Fully Loaded!

Why settle for building and managing a vanilla Web site when you can savor a full featured **Hot Banana** Web site?

Hot Banana is a fully-loaded Web content Management System with all the Web site optimization and eMarketing campaign tools your Marketing department craves. And, it's easy-to-use, search engine friendly and fine tuned for peak performance.

So what's the cherry on top? It's ColdFusion of course - you'll love how easy **Hot Banana** is to implement, integrate and customize.

Schedule your **Free** Taste Trial Today!

Contact Us Now:
hotbanana.com/cfdj-demo
1.866.296.1803
sales@hotbanana.com

Features...

- 100% browser-based
- Updated RTE
- Powerful DAM
- XHTML & CSS compliant
- Multilingual support
- Granular security
- Flexible workflow
- Press release manager
- Keyword analysis
- Web analytics center
- A/B Testing
- Powerful DAM
- Form builder
- RSS & blogs
- SEO tools
- Content reuse & scheduling
- Lead source logging & tracking
- Custom metadata
- Event registration
- Email manager



Web Content Management For Marketing



founder & ceo

Fuat Kircaali fuat@sys-con.com

group publisher

Jeremy Geelan jeremy@sys-con.com

advertising

senior vp, sales & marketing

Carmen Gonzalez carmen@sys-con.com

advertising sales director

Megan Mussa megan@sys-con.com

associate sales manager

Corinna Melcon corinna@sys-con.com

sys-con events

president, events

Carmen Gonzalez carmen@sys-con.com

events manager

Lauren Orsi lauren@sys-con.com

customer relations

circulation service coordinator

Edna Earle Russell edna@sys-con.com

sys-con.com

vp, information systems

Robert Diamond robert@sys-con.com

web designers

Stephen Kilmurray stephen@sys-con.com

Richard Walter richard@sys-con.com

accounting

financial analyst

Joan LaRose joan@sys-con.com

accounts payable

Betty White betty@sys-con.com

subscriptions

Subscribe@sys-con.com

Call 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

Binary Data, ColdFusion & Flex

Sending BitmapData to a server and saving it as a JPG file

By Andrew Trice

Several months ago I posted some articles on my blog about Flex 2 components and accessing/modifying their BitmapData. In one example, I sent the BitmapData to the server and saved it as a JPG file, and I've been asked numerous times since...

"How did you do that?" It's surprisingly easy to do once you understand the concepts involved. There are four ways to get binary data from the Flex application back to your server: AMF3 (RemoteObject), Web Services, HTTP Services, or through a Socket connection. In this article I'll cover the first three topics as they pertain to Flex 2; Socket connectivity could take an article all by itself.

Binary data can't be pushed to the server in its native format using a Web Service or a standard HTTP POST method. To save the data using Web Services or HTTP POST, you must first convert the binary data to a text string using Base64-encoding. On the other hand, AMF3 (RemoteObject method) lets you send the binary data to the server in its native binary form. One thing to keep in mind with Base64-encoding is that the encoding process will actually increase the size of the data that's being sent across the wire.

Regardless of how you're sending the data to the server, it's a good practice to compress the data client side whenever possible. I've used the JPEGEncoder class at <http://code.google.com/p/as3corelib> with great success. You can use this class to convert binary image data into a compressed JPG ByteArray that can be sent to the server. This is a good practice for two reasons:

- The data is compressed, which helps

decrease latency when communicating with the server.

- The data is encoded into the format that you want to save, so no additional processing/conversion is required on the server. You simply need to save the data either in your file system or in a binary object in your database.

Here's how you get data from a Flex component into a JPG ByteArray: First, you'll have to retrieve the BitmapData from your Flex component. You can pass any Flex component into the following function to retrieve its BitmapData:

```
private function getUIComponentBitmapData( target :
UIComponent ) : BitmapData
{
    var bd : BitmapData = new BitmapData( target.width,
target.height );
    var m : Matrix = new Matrix();
    bd.draw( target, m );
    return bd;
}
```

Once you have the BitmapData, you'll have to create an instance of the JPEGEncoder class and encode the BitmapData. (This example uses the JPG quality of 75.) It's also important to remember that your Flex application will have a slight pause while the encoding is being processed:

```
var bd : BitmapData = getUIComponentBitmapData(
paintCanvas );
var encoder : JPEGEncoder = new JPEGEncoder(75);
var data : ByteArray = encoder.encode( bd );
```

Once you have the data converted to a JPG ByteArray, you're ready to push it to the server and save it. The fastest and easiest way to do that is to use a RemoteObject method and serialize the data using AMF3. This example shows you a method in a ColdFusion Component (CFC) that will let you send the data and save it to the local file system:

TABLE OF CONTENTS

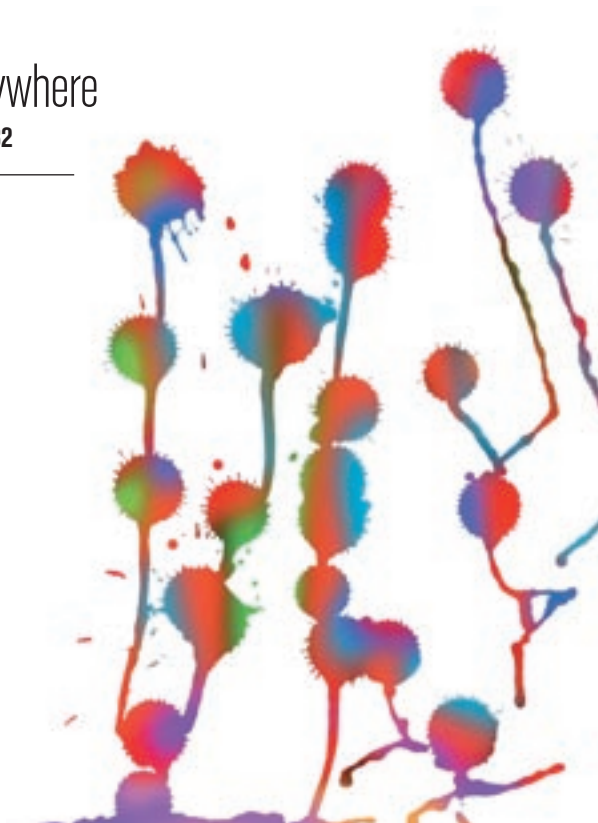
Object-Oriented Form Validation

Matt Osburn...22



Objects Everywhere

By Jeff Chastain...32



5

editorial

Career Growth 101 for the ColdFusion Developer

By Simon Horwith

7

bitmap data

Binary Data, ColdFusion and Flex
Sending BitmapData to a server and saving it as a JPG file

By Andrew Trice

12

components

Inheritance: Code Reuse for CFCs
When it comes to code, less is more

By Josh Adams

20

viewpoint

Building the Right Project Team
Is Key to a Successful Technical Implementation

The rule of five

By Robert Shinbrot

26

cfugs

ColdFusion User Groups

28

cfc

Don't Talk to Strangers!

ColdFusion components
and the Law of Demeter

By Kola Oyediji


```
<cfcomponent name="ImageSave" displayName="ImageSave" output="false">
    <cffunction name="ROsave" access="remote" output="false"
    returntype="void">
        <cfargument name="data" type="binary" required="true" />
        <cffile action="write" file="c:\temp\ro_data.jpg"
output="#arguments.data#" />
    </cffunction>
</cfcomponent>
```

You can see that the code is actually very simple. The CFC's ROsave (remote object save) method is expecting binary data as a parameter. When executed, the data is written to the file system using the <CFFILE /> "write" method.

On the Flex side, we'll have to instantiate a mx:RemoteObject:

```
<mx:RemoteObject
id="ro"
showBusyCursor="true"
destination="ColdFusion"
source="BinaryData.cf.ImageSave">

<mx:method name="ROsave"
result="onResult('Data Saved via mx:RemoteObject')"
fault="onFault(event)" />
</mx:RemoteObject>
```

To save the data, we'll invoke the ROsave method and pass the JPG-encoded ByteArray as a parameter:

```
var bd : BitmapData = getUIComponentBitmapData( paintCanvas );

var encoder : JPEGEncoder = new JPEGEncoder(75);
var data : ByteArray = encoder.encode( bd );

ro.ROsave( data );
```

If you aren't using remoting, you can save the data using Web Services or HTTP services. Most seasoned ColdFusion developers might stop me here and say... "If you're using CFCs as Web Services, why wouldn't you just use them as RemoteObject methods since they are faster?" My response is this: This is just an example. You may be able to take this method and apply it to other technologies where it may be applicable (.NET, Java, PHP, etc.).

```
<cfcomponent name="ImageSave" displayName="ImageSave" output="false">
    <cffunction name="WSsave" access="remote" output="false"
    returntype="void">
        <cfargument name="data" type="string" required="true" />
        <cffile action="write" file="c:\temp\ws_data.jpg"
output="#ToBinary(arguments.data)#" />
    </cffunction>
</cfcomponent>
```

You can see that the code for the Web Service method is

very similar to the previous example. The only difference is that the toBinary method is being used to convert the data from a Base64-encoded string into binary data. The CFC's WSsave (Web Service save) method is expecting a Base64-encoded string as a parameter. When executed, the data is also written to the file system using the <CFFILE /> "write" method.

In Flex, we need an instance of a mx:WebService to save the data:

```
<mx:WebService
id="ws"
showBusyCursor="true"
wsdl="/BinaryData/cf/ImageSave.cfc?wsdl">

<mx:operation name="WSsave"
result="onResult('Data Saved via mx:WebService')"
fault="onFault(event)" />
</mx:WebService>
```

To save the data, we first need to Base64-encode it. The following function will take care of that for us:

```
private function base64Encode( data : ByteArray ) : String
{
    var encoder : Base64Encoder = new Base64Encoder();
    encoder.encodeBytes( data );
    return encoder.flush();
}
```

We'll then invoke the WSsave method and pass the Base64-encoded ByteArray as a parameter:

```
var bd : BitmapData = getUIComponentBitmapData( paintCanvas );

var encoder : JPEGEncoder = new JPEGEncoder(75);
var data : ByteArray = encoder.encode( bd );

ws.WSsave( base64Encode( data ) );
```

If you want to save the binary data without using RemoteObjects or Web Services, you can always use a standard HTTP post method. In Flex, you'll have to create an instance of an HTTPService object, with the method set to "POST":

```
<mx:HTTPService
id="hs"
showBusyCursor="true"
useProxy="false"
method="POST"
resultFormat="text"
url="/BinaryData/cf/HTTPImageSave.cfm"
result="onResult('Data Saved via mx:HTTPService')"
fault="onFault(event)" />
```

— CONTINUED ON PAGE 18

This is a lot easier said than done, but if you have a good idea, the payoff could be substantial. With either of these approaches you also need to put a value on security. Consider yourself blessed if you are employed by and have interest in a company that not only has a good chance at being bought or going public, but that also offers job security. A very large percentage of these companies are at a higher risk of folding and never realizing their dream. Developing your own product or Web site is an even higher risk if you're completely self-employed, and a good approach would be to keep your day job and work on said project on weekends and evenings until you feel comfortable that the project will succeed.

If your expectations aren't so high that you are planning retirement just yet, there are several ways to improve your worth and command a better salary. In today's job market, a ColdFusion developer who knows object-oriented programming concepts and how to apply them in CF is worth quite a bit more and is in more demand than a developer with twice the experience but who doesn't have OO experience. Learn about OOP and master CFCs and the techniques for developing OO CF apps, and emphasize this on your resume. Knowledge and experience with a framework(s) carries weight with some companies and is another thing to highlight on your resume, though most companies that are looking for a "framework developer" tend to accept inexperience with a framework if the hire knows OO. Learning Flex 2 or Java certainly also helps to make you more marketable – in particular the market for CF developers who also know Flex is a rapidly expanding one. ColdFusion Developer Certification doesn't generally carry much weight in my experience, but it does look good on paper and some employers and contracts do require it – and getting certified doesn't take too much time, effort, or money. Aside from having decent knowledge with whatever database an employer is using, most of the other technical skills that set a developer apart from the pack are more related to management. Knowledge and experience with a source control system like subversion tends to carry weight with employers, as does (in fewer circumstances though) experience with Ant for deployment.

Speaking of management, being in a tech/team lead or architect role generally comes with a higher salary, as well as more responsibility. So does project management, though emphasizing project management and/or looking for project management positions does mean that you are committing to a career path that will most likely take you away from day-to-day coding and into a path of management and executive management. If that is a career path that you find appealing, by all means pursue it – there will always be a demand for PMs, and their pay tends to be really good. If you have experience leading/managing projects or development teams, definitely emphasize this on your resume and in your interviews.

If your goals are more focused on quality of life, by which I mean vacation, low stress, etc., it's a bit more difficult to state a formula for achieving your goals. Be aware that working for an employer as an internal employee who builds internal applications certainly does tend to mean more moderate hours and less stress surrounding deadlines than working for a solutions company where you are developing applications for clients.

This is on an employer-by-employer basis, but is a practical piece of advice. Be aware that pay is not always as good in these positions, but again – that's on a case by case basis. Whether your goals are solely financially driven or include aspirations toward public notoriety and other noteworthy achievements, the best way to attain either/both is to work on large, complex applications. Applications that require a large team, that have high public visibility, and/or that require pushing ColdFusion to its limits tend to be the ones that developers learn the most working on, and also carry a lot of weight with employers. Who you've worked with (i.e., being on a development team with a reputable developer) also tends to carry a lot of weight and gives you an outstanding reference. Other credentials that impress employers include community activity – speaking at user groups and conferences, teaching classes, having a popular blog, and writing/tech editing for magazines (CFDJ) and books. When I explained this I was immediately asked why teaching and community activity makes one more desirable to employers since it doesn't really mean you know more. The truth is that I'm not sure (unless the employer values these things) but it does look good on paper and it lends credibility...and for many employers it's a clear sign that you have good written and oral presentation skills...something that many companies look for in their IT staff but is very hard to find. I definitely advise developers seeking to make themselves more marketable to write and present, and if you are offered the opportunity to teach or to become a certified instructor, definitely don't pass up the chance unless you really have no desire to be an instructor.

Many of you may be saying to yourself "that's all fine and good, but I don't have access to all that at my current job, so what should I do?" The answer is quite simple. If you're ready to leave wherever you are currently employed, take a look at the points above and do whatever you can to make yourself more marketable, then begin the job search. If you like where you are but don't have the opportunities to "raise the bar" at work, my advice would be to first see if there are opportunities at work that you're missing. The next time you have the chance, take that leadership role, assist the PM to better understand what he or she does, try coding something in a more object-oriented fashion or in a more advanced manner than usual, mentor the new member of the team, go out of your way to do something with CF that you've never done before, and above all try to enjoy what you're doing. Not thinking of your job as a job, but as a craft – as an interesting and challenging role that only years of strenuous exercise will perfect (and that's fun to do) will motivate you, make you happier and better at your job, and will show around the office as well as in interviews. The truth is that the most successful developers love what they do. If nothing else – get more active in the community. Join CF-Talk and some other list servers, volunteer to give a presentation at your local user group, submit applications to speak at a conference and attend every conference you can whether you're speaking or not, e-mail me with an article idea, even volunteer to contribute to one of the open source projects (or create your own). Just get involved.

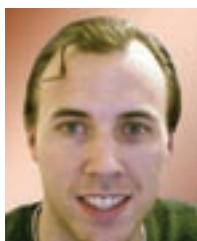
Other companies in this magazine spent a lot of time on pretty ads. As you can see, we did not. We spent our time hiring the best people and training them to deliver outstanding support for your website. We spent our time building a state of the art datacenter and staffing it with people who care about your website like it's their own. Compassion, respect, credibility, ownership, reliability, "never say no," and exceed expectations are words that describe our service philosophy. From the first time you interact with us, you'll see what a difference it really makes. And you'll also forgive us for not having a pretty ad.



WEB HOSTING • MANAGED DEDICATED SERVERS • COLOCATION • VPS • ECOMMERCE • BLOGGING • EMAIL

Inheritance: Code Reuse for CFCs

When it comes to code, less is more



By Josh Adams

When it comes to code, less is more. Or to be more precise, less is

better. How do you write less code and still write enough code to make your applications function as required? One

way is to reuse code: write code once, use that code many times.

If the words “code reuse” evoke thoughts of copying bits of code and pasting it elsewhere, keep reading because this article will teach you a better way!

When it comes to object-oriented ColdFusion Markup Language (CFML) using ColdFusion components (CFCs), one approach to writing code once and using it many times is making use of inheritance. This article will explain what inheritance is and how you can take advantage of it in order to write less code. Along the way, this article will ensure that you understand the basics of CFCs and how they are used.

The best software not only functions as required by its users, but also allows for ease of maintenance. Maintenance is important because if an application is easily maintainable, it can continue to function as required by its users even as those requirements change over time. However, in order to produce functional software in a deadline-driven world, maintainability is often neglected. But is it possible that both the goals of immediate functionality and long-term maintainability can be served simultaneously in the initial development? Yes, and the key is to write as little code as possible to get the job done. As a general rule, less code equals both less bugs (or, put another way, better functionality) and less maintenance, not only with respect to bug fixes, but also in terms of code alteration in response to new or changed functional requirements.

CFML has long allowed for numerous types of code reuse such as including files, calling custom tags, and calling user-defined functions (if you are unfamiliar with these, you should make learning about them a priority). As is true of any CFML application, it is possible to leverage all these types of code reuse to improve the design of a CFML application that makes use of CFCs. However, CFCs can also leverage another type of code reuse: inheritance.

The Basics of CFCs

Since the use of CFCs is central to object-oriented CFML development and since inheritance in CFML is only applicable to CFCs, let's take a moment to look at the basics of CFCs before we dive into the specifics of inheritance. CFCs are special CFML files used in a particular way in CFML applications, yet the contents of a CFC are not so very different from the contents of other CFML files. Instead, the fundamental difference between CFCs and other CFML files lies in their use: CFCs are not used directly as part of a request; instead, CFCs are used as the basis for the creation of memory-resident objects. We will get to how that works in a moment, but before that we need to see what a CFC looks like. This simple example shows the contents of a file named `simple.cfc`:

```
<cfcomponent>
    text output
    <cfset this.myVar = "statement">

    <cffunction name="myMethod">
        <cfreturn "string">
    </cffunction>
</cfcomponent>
```

This simple CFC is largely similar to other CFML files – note that it makes use of only one special CFML tag, the `<cfcomponent>` tag, which consists of an opening tag on the first line of the file and a closing tag on the last line of the file. This tag specifies that the file is a CFC. Although Adobe ColdFusion MX does not strictly require the use of the `<cfcomponent>` tag, good coding practice dictates it be used in CFCs.



Solution
PARTNER

INTERGRAL
information solutions

</cf_bugs>

2.0



FusionDebug™ - The Interactive CF Debugger
Faster. Easier. Smarter.



www.fusiondebug.com

Announcing FusionDebug™ 2.0

- installers for Windows, MacOS, Linux...
- improved user interface layout
- easy to use configuration tool
- improved support for CF frameworks
- step through CF pages, tags and CFC's
- set breakpoints
- view and modify variables
- watch expressions

While the rest of the code in the CFC is familiar CFML, some new terminology is used. User-defined functions in a CFC are called “methods.” Thus, the last three lines of `simple.cfc` before the closing `</cfcomponent>` tag implement the `myMethod` method of `simple.cfc`. Variables in a CFC are called properties. Thus, the third line of `simple.cfc` creates the `myVar` property of `simple.cfc`. Notice the use of the `This` scope in the creation of the `myVar` property. The `This` scope is a special scope for use only in CFCs; it is used to cause a property of a CFC to be accessible not only within the CFC but within code that uses the CFC. This is called making the property public. If instead in the `Variables` scope of the CFC is used, the property is made private, meaning that it is accessible only within the CFC and not from code that uses the CFC. Methods too can be made public or private via use of the `access` attribute of the `<cffunction>` tag; when this attribute is omitted, as it was in the method definition for `myMethod` in `simple.cfc` above, the method is public.

Now we can look at the way CFCs are used. This simple example provides the contents of a file named `callsimple.cfm`:

```
<cfset myObj = CreateObject("component", "simple")>
<cfoutput>
  <br>
  #myObj.myVar#<br>
  #myObj.myMethod()#<br>
</cfoutput>
```

Note that in this example, the `myObj` variable references in `callsimple.cfm` were left unscoped for the sake of simplicity. Good coding practice dictates that variable references be scoped.

As mentioned previously, CFCs are not used directly as part of a request. That means they are not called directly from a browser nor are they included with `<cfinclude>` nor are they even called with `<cfmodule>`. Instead, as demonstrated on line 1 of `callsimple.cfm`, CFCs are used with the `CreateObject` function (you may also use the `<cfobject>` tag; its functionality is the same as that of the `CreateObject` function and the choice of which to use is largely a personal preference). When “component” is the value of the first argument passed to the `CreateObject` function, an instance is created of the CFC specified by name in the second argument passed to the `CreateObject` function. An instance of a CFC is a memory-resident version of the CFC that contains all properties and methods defined in the CFC. An instance of a CFC is also called object. A theoretically unlimited number of objects can be created from any CFC.

But how are a CFC’s properties and methods created? When the instance of the CFC is created, the “pseudo-constructor” of the CFC is called. Effectively, this means that all the code in the CFC that defines methods is used to create the object’s methods and that all other code in the CFC, including any code that creates the object’s properties, is executed.

Once an object has been created from a CFC, it and its public properties and methods can be referenced. By assigning the object to a variable, it and its public properties and methods can be referenced an unlimited number of times during the life of that variable. In line 1 of `callsimple.cfm`, we see that the object created from `simple.cfc` is assigned to a variable named `myObj`. Because no scope is declared for `myObj`, it is placed in

the `Variables` scope. On the fourth line of `callsimple.cfm`, we see the public `myVar` property of the `myObj` instance of `simple.cfc` being referenced, and on the fifth line of `callsimple.cfm`, we see the public `myMethod` method of that same `myObj` instance of `simple.cfc` being called. Referencing an object’s public properties and methods is as simple as using the variable name to which the object was assigned followed by a period (a.k.a. “dot”) and then the name of the property or method.

When calling a method of an object, we also must include a pair of parentheses after the method name. Inside those parentheses we must include, if any for the given method, a comma-delimited list of the arguments to be passed to the method. There are other ways to pass arguments to methods, but for simplicity’s sake we won’t cover those in this article.

Now that we know what is going on in `callsimple.cfm`, let’s look at its output:

```
text output
statement
string
```

The first line of the output results from simply creating an instance of `simple.cfc` on the first line of `callsimple.cfm` and the resultant pseudo-constructor execution of all code in `simple.cfc`. The second line of the output results from referencing the public `myVar` property of `myObj` on the fourth line of `callsimple.cfm`. Finally, the third line of the output results from referencing the public `myMethod` method of `myObj` on the fifth line of `callsimple.cfm`.

With a basic understanding of CFCs in place, let’s move on to the purpose of this article: reusing code with CFCs. As mentioned earlier, we can reuse code in CFCs not only via all the standard CFML code reuse methods – including files, calling custom tags, and calling user-defined functions – but also via inheritance. But what exactly is inheritance?

CFCs and Inheritance

In the real world, inheritance involves one entity gaining things as the result of a relationship with another entity. Inheritance in CFCs is very similar: one CFC gains things (specifically, CFML code) as the result of a relationship with another CFC. We call a CFC that inherits from another a “child CFC” and we call the CFC from which that child inherits a “parent CFC.” CFML inheritance is virtual, meaning both the child CFC and the parent CFC contain the complete CFML code of the parent CFC, without duplication of the CFML code of the parent CFC in the child CFC file.

Why is inheritance useful? Because it allows us to write code in one CFC and then use that code in numerous other CFCs. Thus, we can write that CFML code one time and have it virtually contained in an unlimited number of CFCs through inheritance. By writing that CFML only one time, it’s less likely we will write bugs into it and, should a bug be discovered in it or should a requirement for its functionality change, we need only modify a single instance of the code.

Now that we have an understanding of CFC inheritance, let’s take a look at how it’s used in a file called `exsimple.cfc`:

```
<cfcomponent extends="simple">
```




Adobe



CFDynamics



ColdFusion Developer's Journal

cf^{united}

June 27- 30, 2007

Washington D.C area



Step through your CFML code with FusionDebug Using Structures with CFCs
ColdFusion Application Security CSS - Back to Basics Flex and Ajax: perfect match
 Object Oriented **Flex** Testing, Monitoring & Tuning CF w/ Open Source tools
 Testing for Accessibility Coding with XML **Flex 2** for ColdFusion Developers
 Flex Data Services & CF **Scorpio** - Faster, Better & Cooler! Custom Taglibs for CFCs & CFCs
 AJAX Development with ColdFusion Frameworks Programming with ColdFusion Spring
 Integrating Spry and ColdFusion Creating and Consuming Web Services
 Beyond Basic **SQL** For CF Working with RSS
Flex 2 for ColdFusion Developers **ColdFusion** Integrating Spry
 Programming with ColdSpring Integrating Spry
 Creating and Consuming WebServices Using Structures
 CSS - Back to Basics Flex Data Services & CF **Scorpio**
 Coding with XML Step through your CFML code with
ColdFusion Application Security Object
 Flex and Ajax: perfect match Using Structures with CFCs
 Testing, Monitoring & Tuning CF w/ Open Source tools Beyond



TeraTech

HostMySite.com



Paper | Thin



Your seat

the premier coldfusion technical conference

Don't miss out on the Premier ColdFusion conference, CFUNITED-07 where you can learn from over 50 expert speakers and authors such as Ben Forta, Michael Smith, Simon Horwith, Hal Helms, Ray Camden, Charlie Arehart. Four days of sessions, networking and learning with your programming peers from around the world. Guarantee yourself a seat and register today.



"CFUNITED is a great way to not only network with fellow ColdFusion aficionados, but also to learn new techniques and methods to help your career or business. You also can get inside scoops on developments in ColdFusion and learn best practices and much more."
 -Angela T, Attendee

"It is run by developers for developers. The sessions are based on real world experience and real user case studies."
 -Rich P, Attendee

"CFUNITED is a great opportunity to launch your career into the next level, allowing developers to learn new tips, tricks and techniques, all while bonding with fellow ColdFusion users many of us only know by name."
 -Constanty D, Attendee

"Learning more, networking more, getting NEW ideas...and now, to support CF's continued existence."
 -Howard P, Attendee



TeraTech Inc.
 405 East Gude Dr, Ste 207
 Rockville, MD 20850
 301.424.3903

www.cf^{united}.com

```
<cffunction name="newMethod">
    <cfreturn "newstring">
</cffunction>
</cfcomponent>
```

Note the difference between the opening `<cfcomponent>` tag of `simple.cfc` and the opening `<cfcomponent>` tag of `exsimple.cfc` – `exsimple.cfc` has `extends="simple"` in that opening `<cfcomponent>` tag. The `extends` attribute of a CFC's `<cfcomponent>` tag is used to specify that CFC's parent. As such, by using `extends="simple"` in the opening `<cfcomponent>` tag, we cause `exsimple.cfc` to virtually contain all the CFML in `simple.cfc`. Thus, when an instance of `exsimple.cfc` is created with either the `CreateObject()` function or the `<cfobject>` tag, the pseudo-constructor of `simple.cfc` is called, causing the CFML code contained in `simple.cfc` to be executed, just as if that code were located in `exsimple.cfc`. In addition, after an instance of `exsimple.cfc` has been created, we can reference on that instance any of the public properties and methods defined in `simple.cfc` just as if they were defined in `exsimple.cfc`. Let's see what this looks like in a file named `callexsimple.cfm`:

```
<cfset myObj = CreateObject("component", "exsimple")>
<cfoutput>
    <br>
    #myObj.myVar#<br>
    #myObj.myMethod()#<br>
    #myObj.newMethod()#<br>
</cfoutput>
```

We create an instance of `exsimple.cfc`. Because `exsimple.cfc` extends `simple.cfc`, we can reference the public `myVar` property and the public `myMethod` method of `simple.cfc` as well as the public `newMethod` method of `exsimple.cfc`. The output of `callexsimple.cfm` is exactly what we expect:

```
text output
statement
string
newstring
```

Now if we update `simple.cfc`, `exsimple.cfc` will automatically get the benefits of that change without being changed at all. Because we didn't have to duplicate code from `simple.cfc` in order to use it in `exsimple.cfc`, we were able to write less code. Less code is better code because it gives us fewer places to introduce bugs and because it's easier to maintain.

Multiple Inheritance

So far we have only considered inheriting into a child CFC a single parent CFC via the `extends` attribute of the `<cfcomponent>` tag. Can we inherit multiple CFCs by using the `extends` attribute of the `<cfcomponent>` tag? The answer is very intentionally no; multiple inheritance has certain complications (a discussion of these is beyond the scope of this article) and so a deliberate decision was made to allow for only single inheritance in CFML. However, there is a new type of limited multiple inheritance that can be used in CFML deployed via New Atlanta's BlueDragon 7.0: interfaces. An interface is a special

type of CFC that cannot be directly instantiated and that can only specify, but not implement, methods. An interface cannot contain any other pseudo-constructor code, including statements that create object properties. An unlimited number of CFC interfaces can be inherited in any CFC by using BlueDragon 7.0's new implements attribute of the `<cfcomponent>` tag. Such inheritance does not gain a child CFC any new functionality since the methods of the interface are not implemented. What is gained by the child CFC is the requirement that it implement all methods specified in the interfaces it implements. In a sense, an interface defines a contract that any CFC implementing the interface must follow. In order to understand why this is useful, we have to understand another important concept: type matching.

Implementing type matching in CFML code is another way to make it more reliable and maintainable. Type matching involves explicitly specifying how the code is intended to be used, which makes the code easier to read, easier to use properly, and easier to maintain. In addition, an explicit specification of how the code is intended to be used makes it more likely that improper use of the code will be identified sooner rather than later, either via the generation of errors in testing or by inspection of the code. For example, when we define a CFML function/method with the `<cffunction>` tag, we may use the `returnType` attribute to specify the type of data that is returned by the function/method. In the same way, when we define a CFML function/method argument with the `<cfargument>` tag, we may use the `type` attribute to specify the type of the data that must be supplied for that argument. For either of these attributes, if the specified type is not passed, an error is generated. In addition, for either of these attributes, we may specify the name of a CFC and by doing so we are indicating that the data being passed must be an instance of the specified CFC.

What does type matching with CFC types gain us? It gains us the ability to reliably reference the public properties and methods the specified type defines. In other words, if we specify `'type="simple"'` in a `<cfargument>` tag, we can reference the `myMethod` method inside the function to which that `<cfargument>` tag applies because that method is defined in `simple.cfc`. Here is where inheritance comes in: because a child CFC inherits all of the properties and methods of its parent CFC, the public properties and methods of the parent CFC can be reliably referenced on an instance of the child CFC. That is, if we specify `'type="exsimple"'` in a `<cfargument>` tag, we can reference the `myMethod` method inside the function to which that `<cfargument>` applies because that method is defined in `simple.cfc` and `exsimple.cfc` inherits `simple.cfc`. As a result of this behavior, an instance of a child CFC matches the type of its parent CFC and, for that matter, the type of any CFC from which that parent CFC inherits, and any CFC from which that CFC inherits, and so on all the way on up the inheritance chain. It is this very fact that type matching honors inheritance that explains the usefulness of an interface: since an interface is a type that can be inherited, when an object matches the type of an interface, that interface's properties and methods can be referenced reliably on that object.

We have standard CFCs that support standard single inheritance and interfaces that allow for a special type of multiple inheritance that is useful for type matching. But what about something that is between a standard CFC and an interface?



Register Now!
**EARLY-BIRD
SAVINGS!**
\$200

**ROOSEVELT HOTEL
NEWYORK CITY**

June 25-27, 2007

Virtualization: Solutions for the Enterprise.

Delivering The #1 i-Technology Educational and Networking Event of the Year!

As today's CIOs and IT managers rise to the challenge of using their enterprise computing infrastructure as cost-effectively as possible while remaining responsive in supporting new business initiatives and flexible in adapting to organizational changes, Virtualization has become more and more important.

A fundamental technological innovation that allows skilled IT managers to deploy creative solutions to such business challenges, Virtualization is a methodology whose time has come. The fast-emerging age of Grid Computing is enabling the virtualization of distributed computing, of IT resources such as storage, bandwidth, and CPU cycles.

But Virtualization can also apply to a range of system layers, including hardware-level virtualization, operating system level virtualization, and high-level language virtual machines.

Register Online!

www.VirtualizationConference.com



THE FIRST MAJOR VIRTUALIZATION EVENT IN THE WORLD!

**HURRY, FOR EXHIBITOR AND
SPONSORSHIP OPPORTUNITIES
CALL 201-802-3020**

*Learn from
the Experts...*



— BROUGHT TO YOU BY —



BlueDragon 7.0 provides that, too, and it's called an abstract CFC. Like a standard CFC, an abstract CFC allows for the implementation of properties and methods, and like a standard CFC, an abstract CFC may only inherit from a single other CFC. However, like a CFC interface, an abstract CFC also allows for the specification of methods that must be implemented by any CFC that inherits from that abstract CFC, and like a CFC interface, an abstract CFC cannot be instantiated directly. An abstract CFC is a CFC that is not a completed CFC implementation; it is a partial CFC implementation designed specifically to be inherited by CFCs that complete that partial implementation.

Conclusion

We have now seen how inheritance allows us to minimize the amount of CFML we have to write when working with CFCs. Any CFC can inherit the full CFML contents of a single other CFC by extending that CFC. With BlueDragon 7.0, we can use abstract CFCs, a special type of CFC that can be extended but not directly instantiated. In addition, we have seen that with BlueDragon 7.0, a CFC can inherit unimplemented methods from an unlimited number of CFC interfaces, which is useful for CFC type matching.

In order not to detract from the focus of this article, certain object-oriented considerations such as overriding, abstraction, encapsulation (including the use of getter and setter methods), and polymorphism intentionally were not covered and certain statements impacted by such considerations intentionally were not fully qualified.

Resources


If you are interested in a fuller introductory treatment of object-oriented development with CFCs than I was able to give in this article, see the following presentation by Matt Woodward:

http://www.mattwoodward.com/presentations/cfc_101.pdf

If you are interested in more information on using inheritance as well as on aggregation (a concept not discussed in this article), see Hal Helms's newsletter from March 22, 2003:

http://www.halhelms.com?fuseaction=newsletters.show&issue=032203_Inheritance

Finally, if you are interested in more information on CFC interfaces and abstract CFCs as supported in BlueDragon 7.0 see the BlueDragon 7.0 cfml Enhancement Guide:

http://www.newatlanta.com/products/bluedragon/self-help/docs/7_0/BlueDragon_70_CFML_Enhancements_Guide.pdf 

About the Author

Josh Adams is the developer evangelist for New Atlanta's BlueDragon family of CFML application server products. He presents on a regular basis at technical conferences and user groups throughout North America. Josh is also an active CFML developer both in his role at New Atlanta and in other endeavors.

Binary Data, ColdFusion & Flex

— CONTINUED FROM PAGE 9

The file HTTPImageSave.cfm is actually very simple. The save occurs with only two lines of code:

```
<cfparam name="data" type="string" default="">
<cffile action="write" file="c:\temp\http_data.jpg" output="#ToBinary( data )#" />
```


You'll notice in this case that the data is also being written to the file system using the toBinary method (to convert the Base64-encoded string back into binary data). When invoking the HTTPService, you'll have to create an object containing the parameters for the HTTPService and send it as the parameter of the HTTPService.send method (note "hs" is the id of the HTTPService instance):

```
var bd : BitmapData = getUIComponentBitmapData( paintCanvas );

var encoder : JPEGEncoder = new JPEGEncoder(75);
var data : ByteArray = encoder.encode( bd );
var params : Object = { data : base64Encode( data ) };
hs.send( params );
```

...and that is how you save binary image data using RemoteObjects, Web Services, or HTTPServices. This doesn't just apply to images. This applies to any kind of binary data; the only difference is that the image data first gets encoded to a JPG ByteArray. The AS3 corelib project on Adobe Labs/Google Code also includes class libraries that enable you to save data as PNG images instead of JPG images, so JPG isn't your only option.

A few things to keep in mind for real-world applications... It's best practice to use <CFFILE /> in a <CFTRY/> statement to catch any file system errors that may occur (not enough space, permissions, etc.). It's also best practice to use <CFLOCK /> with <CFFILE /> to prevent any errors due to data synchronization, threading access, or deadlock scenarios.

A working example and the source code from this article can be viewed online at http://www.cynergysystems.com/blogs/blogs/andrew.trice/binary_data_example/ or downloaded from http://www.cynergysystems.com/blogs/blogs/andrew.trice/binary_data_example/BinaryData.zip. 

About the Author

Andrew Trice is a consultant with Cynergy Systems in Washington, DC, where he specializes in development of Flex-based Rich Internet Applications. Andrew has over 5 years of proven experience in the RIA industry, including application design and development using Flex, Flash, ColdFusion, J2EE and .NET architectures.

andrew.trice@cynergysystems.com



See What's New at the 2006 JavaOneSM Conference

Come to the 11th Annual JavaOneSM conference and see where the JavaTM platform is headed. More than 300 technical sessions and BOFs equip you with the knowledge you need for your current and future technological innovations.

Hear from industry experts, including Graham Hamilton and Bill Shannon of Sun Microsystems, as they discuss key directions for the next releases of the Java platform.

Plus, enhance your development skills in hundreds of expert-led sessions in five tracks over four days:

- : Track One :
Java Platform, Standard Edition (Java SE)
- : Track Two :
Java Platform, Enterprise Edition (Java EE)
- : Track Three :
Java Platform, Micro Edition (Java ME)
- : Track Four :
Tools
- : Track Five :
Cool Stuff

GET CONNECTED

with the JavaOne Conference Event Connect tool.

As a Conference attendee, you can now connect with experts and fellow developers before the event. This effective online solution allows you to quickly schedule meetings with industry leaders and interact with other attendees.

Register for the Conference and get connected today.

REFER YOUR FRIENDS

and receive an iPod nano music player!

One iPod nano music player per qualifying registrant. While supplies last.

Last Chance to **SAVE \$200!**
REGISTER
by April 14, 2006, at java.sun.com/javaone/sf

**Content subject to change.*

PLATINUM COSPONSORS



GOLD COSPONSORS

SILVER COSPONSORS



JavaOneSM Conference | May 16–19, 2006
JavaOneSM Pavilion: May 16–18, 2006, Moscone Center, San Francisco, CA

JavaOne
Sun's 2006 Worldwide Java Developer Conference

Copyright © 2006 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, Java, the Java Coffee Cup logo, JavaOne, the JavaOne logo, Java Developer Conference, Java Community Process, JCP, 100% Pure Java, J2EE, J2ME, J2SE, Jini, Solaris, "Write Once, Run Anywhere," and all Java-based marks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Building the Right Project Team Is Key to a Successful Technical Implementation

The rule of five

By Robert Shinbrot

When building the right project team to complete a custom solution there are many forces at work. These include business drivers, technical drivers, and organizational and political motivations. Regardless of the business or organization there are three basic rules to follow in building a team to deliver a technical solution. The first is to involve the business before the team is even assembled.

Each organization has certain technology standards that govern specific tools and products that can be used on a given project. These standards need to be considered and coordinated within “governance” management when architecting the solution. The third is the driving element that will let you successfully implement any medium or large-scale project and that’s to follow “The Rule of Five.” The Rule of Five is the basis for choosing the right number of people to be on your project team, and if you follow this rule your team will deliver the project on time and on budget. =

Assume you are the project manager for a newly selected technical implementation for a specific line of business that you’re familiar with. You’ve been chosen for this effort because of the confidence that both the business and technology departments have in you to get the job done. Your past record speaks for itself and you have an opportunity to select people currently in your company as well as augment the team with new hires and consultants.

Whether your project requires a medium or large-scale project team you should plan your team accordingly. Always plan in five-person team increments.

1. **Business/Technical Lead** – This person should be someone who understands the business requirements very well and hopefully was one of the main authors of the “Business Requirements Document.” This individual should be technically very strong, but doesn’t necessarily know all the technologies that have to be deployed to implement the system.
2. **Technical Architect** – This person is responsible for designing the technical framework in which the entire system will be built. He should be intimately familiar with all the technologies required to deliver the system and be mindful of all governance requirements in your organization. This is the lead person who will insure the technical success of the project.
3. **Data Analyst** – This person should be knowledgeable about all of the data elements required for system implementation as well as where the data currently resides in the organization and how to gain access to it. This person will coordinate all DBA requirements and work with the governance group in the organization as well as lead all logical and physical database design efforts.
4. **Technical Programmer** – This person will be responsible for coding parts of the system based on the direction of the Project Manager. For example, this person may be the front-end technical programmer.
5. **Technical Programmer** – This person will be responsible for coding parts of the system based on the direction of the Project Manager. For example, this person may be the application server programmer.

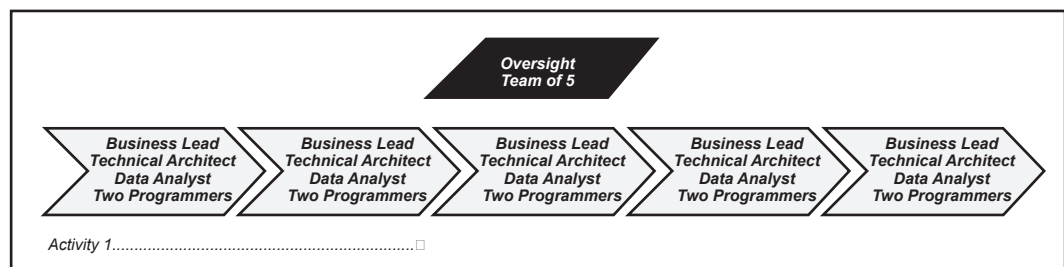


Figure 1: The rule of five


The Rule of Five focuses on any medium or large-scale project that will have a team composition based on these outlines. For example if the project is scoped to be a medium-sized project the team composition is one project manager and a team of five. If the project is scoped to be a large-scale project the team composition is one project manager and n(team of five) or either a 10-, 15-, or a 20-person team. It's rare that a project team is larger than 20, but the same rule holds. Keeping the team composition as listed above allows each team of five to work successfully on their portion of the project.

If your project team is greater than 20 people, the Rule of Five means a team of five to oversee all project activities and provide centralized project coordination or project governance as shown in Figure 1.

When determining who will be on your Rule of Five team follow the basic guidelines. The Business Technical Lead must be someone that is very senior, has direct contact with the business, and can resolve any outstanding business issues that come up. You should handpick this person from a small list of applicants. The Technical Archi-



tect must be very senior and preferably someone you've worked with before. He should have demonstrated superior knowledge in all technical aspects of the project and be hands-on at all times. The Data Analyst should be knowledgeable about ER tools and the SQL language being used in the project. This person should have worked on other projects in this group before to reduce the learning curve. This role is generally overlooked until late in the project. The Technical Programmers tend to be junior compared to other members of the team, but are focused on coding the application.

When building your next project team think in terms of five and you'll be able to maximize your business and technical capability to deliver a solution on time and on budget. A team that's too small or too big will either deliver the project on time but over budget or late and over budget. See if the "Rule of Five" works for you. 

About the Author

Robert Shinbrot has managed very large and complex projects over the last 20+ years within the Financial Services Community. Robert has led the Financial Services Consulting Practice first at Oracle and most recently at BusinessEdge Solutions for the last 5 years.

rshinbrot@businessedge.com


Efficient Web Content Management

CommonSpot[™] Content Server is the ColdFusion developer's leading choice for efficient Web content management.

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

CommonSpot's open architecture and extensive APIs enable easy customization and integration of external applications. With CommonSpot, you can design and build exactly what you need. Best of all, CommonSpot puts content management in the hands of content owners. With non-technical users responsible for creating and managing Web content, developers are freed to focus on strategic application development.

Evaluate CommonSpot today.
To see your site *running* under CommonSpot, call us at 1.800.940.3087.



fast
easy
affordable

features.

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

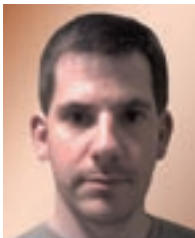
1.800.940.3087
www.paperthin.com

Paper | Thin

© Copyright 2005 PaperThin, Inc. All rights reserved.

Object-Oriented Form Validation

Validating input with self-contained, reusable objects



By Matt Osburn

For Web developers, form validation is a basic task, and yet one that presents some interesting challenges. Even discounting security issues such as SQL injection attacks and cross-site scripting (XSS), form validation is a critical step in your error handling strategy.

The two criteria I use to plan a form validation approach are extensibility and simplicity. Forms will change over time. It's like death and taxes, and much like those two events it's a lot better to plan for them before the fact, rather than after. When your form changes, how easy will it be to change your validation? When your business rules change, as they often do, how easy will it be to update your validation? How easy will it be for someone else to update your validation code if they were not a part of the original development effort? Fortunately, an object-oriented approach handles all of these questions.

Data validation is really two different activities. First, you need

to validate the data. Is a date of birth really a date? Does that credit card number follow the Mod10 algorithm? Second, you need to determine how the data fits into your business rules. Suppose a date of birth of 03/06/1902 was passed in. It's certainly a valid date, but if the point of your application is to sell life insurance policies, then that date will likely not pass your business rules.

Basic data validation and validation of business rules are two different activities, and yet they are so often handled as one. For example, how often have you seen something like this:

```
<cfif not Isdate(form.dob)>
  <cfset errormsg = "Please enter a valid date of birth">
</cfif>
<cfif Year(form.date) GT 1950>
  <cfset errormsg = "Applicant is not eligible for this insurance
  policy">
</cfif>
</cfif>
```

Except for the gross oversimplification of age checking, done for example purposes, this approach is workable, but can be bulky when applied to a large form. Worse, as the business logic is embedded in the datatype validation, the code for the business rule validation must be re-written in order to be used elsewhere.

Thankfully, there's a better way to handle this. To examine a more flexible, reusable approach, let's consider a very simple form. We're going to collect first and last name, date of birth, and country of citizenship, and then we'll validate the input both for correct datatype and business rules.

Validation Bean

Before we start checking for errors, we need some method of collecting errors and error messages. It is so annoying when a Web application tells you that your input was incorrect in some way, but will not tell you what exactly was incorrect or why. Finding errors is only half of the solution; reporting them to the user is the other half. To that end, I use a ValidationBean object (see Listing 1). Using an object to hold validation information creates a standard interface for validation tasks.

The ValidationBean has two properties: Result and Message. Result is a simple Boolean, indicating whether or not the form data passed validation. The Message property is an array of error messages. Along with the necessary getters and setters for these properties, the ValidationBean also has an appendMessage() method, which adds another error message to the Message array, and a Reset() method, which simply clears the error messages and sets the Result to True. I use this method during validation to clear out any results from previous form submissions.

Datatype Validation

The ValidationBean is extremely generic, so at this point you're probably wondering where and how the datatype validation takes place. I put datatype validation within the bean(s) that hold form data. The idea behind this is that the bean should be somewhat self-aware. A bean doesn't need to know how its data is going to be used, but it should have some idea of what its data is and the format for that data.

As you look at the code for formBean.cfc (see Listing 2), a couple of things may stick out. First of all, the setters have an argument with a type of "any" and the getters have a returntype of "any".

Second, all arguments are optional and have a default value, which means that all variables-scoped variables are assured to be assigned a value. This may require some additional explanation, because it may seem better to set the argument types and the returntypes to the datatype that is expected. I find this approach limiting, however. The form bean is a representation of the form data as it was entered. If an incorrect type of data is entered, or if a form control is left blank, I don't want the bean throwing an error. I prefer to find and deal with the problem programmatically.

To handle validation, I include a Validate() method. I pass in a Validation Bean as an argument, and since ColdFusion passes CFCs by reference, it is not necessary for me to return anything from this method. The method can inspect each attribute in the bean and return an error message customized to the form control in question.

At this point what we have is an object that holds the data passed in through a form. This object has an interface for returning data, and the ability to inspect the data it holds to make sure that it is of the expected type. However, it doesn't know how this data is to be used. This is useful if you want to use this form data in several different applications. Consider the example above.

Name, address, date of birth and the like are common pieces of information to pass through a form. Using a form bean to hold and validate data commonly passed in through a form, you don't have to duplicate work on other applications that use this same data. Validation rules are consistent from application to application, and you have the advantage of using an object whose behavior is well tested and well known.

Business Rule Validation

Validating input against business rules is generally more in depth than validating data types. It helps to think of this as giving context to your data. A name, address, and date of birth are nothing more than a set of unrelated data. Once you have applied business rules, however, you have a customer. If business rules give your data meaning, business rule validation ensures that your data is given the correct meaning.

Datatype validation objects are much easier to share than business rule validation objects. This sort of validation is very specific to a given task or application. However, since we are breaking validation into two parts, we can rewrite our business rule validation objects as necessary and reuse our datatype validation objects. Object-oriented applications are really nothing more than a group of objects that are communicating with each other. You, the developer, are free to determine which objects will be a part of a given application. Since our validation objects are loosely coupled, meaning one object doesn't rely on another, we can change out objects as we see fit.

As an example, let's assume that in this particular application our customers need to be between the ages of 13 and 18, and they also need to be citizens of the United States. I would create a Validate() method, perhaps in a customer.cfc object, to check to make sure that the birthdate and citizenship fit the specified parameters. Although this has some similarities to our Validate() method above, the differences are worth pointing out. When validating datatypes, the formBean has no idea how the data will be used. It just knows which form the data is to take. A Customer object, however, is not so much concerned with the representation of the data, but whether or not the data presented can be used to create a valid customer for this application.

Putting It All Together

How you put this all together is really up to you and how you construct your applications. Are you a procedural developer who includes a validation template on a form page once a form submission has been detected? No problem. Your validation template might look something like this:

```
<cfset datatype = CreateObject("component", "myApp.formBean").init(argumentcollection=form)>
<cfset customer = CreateObject("component", "myApp.customer").init()>
<cfset validationBean = CreateObject("component", "myApp.validationBean").init()>

<cfset datatype.Validate(validationBean)>
<cfset customer.Validate(validationBean)>

<cfif ArrayLen(validationBean.getMessages())>
    <!-- Do something to display the error messages -->
<cfelse>
```




```
<!-- No errors. Continue the application -->
</cff>
```

Since I use Mach-II for most of my development needs, I would have an event-bean declared in the mach-ii.xml configuration file for this event, and I would use that bean's Validate() method in an event filter, which is where I would also create my validationBean and, if necessary, any object needed for business rule validation.

Conclusion

Form validation isn't one of those tasks that I look forward to in the morning. Not that I particularly dislike it, but it's just one of those chores of Web development. The object then (no pun intended!) is to make form validation as easy, flexible, and reusable as possible without sacrificing accuracy or functionality.

LISTING 1

```
<cfcomponent name="validationBean" output="false">
  <!-- Properties -->
  <cfset variables.Result = "true">
  <cfset variables.Message = #ArrayNew(1)#

  <cffunction name="init" access="public" returntype="validationBean"
output="false">
  <cfreturn this />
</cffunction>

  <cffunction name="setResult" access="public" returntype="void"
output="false">
  <cfargument name="Result" type="boolean" required="true" />

  <cfset variables.Result = arguments.Result>
</cffunction>

  <cffunction name="setMessage" access="public" returntype="void"
output="false">
  <cfargument name="Message" type="string" required="true" />

  <cfset ArrayClear(variables.Message)>

  <cfset appendMessage(arguments.Message)>
</cffunction>


  <cffunction name="getResult" access="public" returntype="boolean"
output="false">
  <cfreturn variables.Result />
</cffunction>

  <cffunction name="appendMessage" access="public" returntype="void"
output="false">
  <cfargument name="ErrorMsg" default="" type="string"
required="true" />

  <cfset ArrayAppend(variables.Message, arguments.ErrorMsg)>

</cffunction>

  <cffunction name="getMessage" access="public" returntype="Array"
output="false">
  <cfreturn variables.Message />
</cffunction>
```

This is one of the strengths of object-oriented programming. Rather than building one monolithic, interdependent application, it's more like building a house with Legos. This allows you to spend less time on development chores, like form validation, and more time deciding how the Legos should fit together for your application. 

About the Author

Matt Osburn is a Certified ColdFusion developer for Herff Jones, Inc., located in Indianapolis, IN. He has been working as a Web developer for 7 years, specializing in ColdFusion, XML, and AJAX. He can be contacted via e-mail or through his blog at <http://www.pteradactylcry.com>.

cfgeek@gmail.com

LISTING 2

```
<cfcomponent name="formBean" output="false">

  <cffunction name="init" access="public" returntype="formBean">
  <cfargument name="FirstName" default="" required="false" type="any"
/>
  <cfargument name="LastName" default="" required="false" type="any"
/>
  <cfargument name="DOB" default="" required="false" type="any" />
  <cfargument name="Citizenship" default="" required="false"
type="any" />

  <cfscript>
    setFirstName(arguments.FirstName);
    setLastName(arguments.LastName);
    setDOB(arguments.DOB);
    setCitizenship(arguments.Citizenship);
  </cfscript>

  <cfreturn this />
</cffunction>

  <cffunction name="getFirstName" access="public" returntype="Any"
output="false">
  <cfreturn variables.FirstName />
</cffunction>

  <cffunction name="setFirstName" access="public" returntype="void"
output="false">
  <cfargument name="FirstName" required="true" type="Any" />
  <cfset variables.FirstName = arguments.FirstName />
</cffunction>

  <cffunction name="getLastName" access="public" returntype="Any"
output="false">
```

```

    <cfreturn variables.LastName />
</cffunction>

<cffunction name="setLastName" access="public" returntype="void"
output="false">
    <cfargument name="LastName" required="true" type="Any" />
    <cfset variables.LastName = arguments.LastName />
</cffunction>

<cffunction name="getDOB" access="public" returntype="Any"
output="false">
    <cfreturn variables.DOB />
</cffunction>

<cffunction name="setDOB" access="public" returntype="void"
output="false">
    <cfargument name="DOB" required="true" type="Any" />
    <cfset variables.DOB = arguments.DOB />
</cffunction>

<cffunction name="getCitizenship" access="public" returntype="Any"
output="false">
    <cfreturn variables.Citizenship />
</cffunction>

<cffunction name="setCitizenship" access="public" returntype="void"
output="false">
    <cfargument name="Citizenship" required="true" type="Any" />
    <cfset variables.Citizenship = arguments.Citizenship />
</cffunction>

<cffunction name="validate" access="public" returntype="void"
output="false">
    <cfargument name="validationBean" required="true" type="photoonline
ordering.model.beans.validationBean">

    <cff NOT Len(Trim(getFirstName()))>
        <cfset arguments.validationBean.setResult("False")>
        <cfset arguments.validationBean.appendMessage("Please enter your
first name")>
    </cff>
    <cff NOT Len(Trim(getLastName()))>
        <cfset arguments.validationBean.setResult("False")>
        <cfset arguments.validationBean.appendMessage("Please enter your
Last name")>
    </cff>
    <cff NOT IsDate(getDOB())>
        <cfset arguments.validationBean.setResult("False")>
        <cfset arguments.validationBean.appendMessage("DOB must be a
date")>
    </cff>
    <cff NOT Len(Trim(getCitizenship()))>
        <cfset arguments.validationBean.setResult("False")>
        <cfset arguments.validationBean.appendMessage("Please enter your
Country of Citizenship")>
    </cff>

</cffunction>

</cfcomponent>

```

Download the Code...
Go to <http://coldfusion.sys-con.com>

Have you ever
wanted to "see"
what's going on
inside your
ColdFusion servers
and applications?

Now you can.



Seefusion is a unique, powerful tool for real-time monitoring of your ColdFusion applications and servers, providing a wide variety of metrics on request times, database queries, memory utilization, code performance and more!


 **www.seefusion.com**

Seefusion is a product of Webapper Services, LLC
Development, Consulting, Products, Training
www.webapper.com

Career Growth 101 for the ColdFusion Developer

— CONTINUED FROM PAGE 10

To those of you who are interested in bettering your career and aren't committed to staying with the same employer you're currently with, the same rules apply but you also have the added benefit of being able to begin interviewing with companies. Hopefully, my advice here will help you better understand what it is you should look for in a company as well as in an offer. Assuming you have all the skills and credentials to get the job, there are still two things that many developers forget to do when going after work. The first is reviewing their resume — keep it concise and focused. When you're being evaluated for a CF developer position, nobody really cares if you know Quick Basic. Second, remember that attitude is everything, so be confident, have fun, and have a positive attitude about everything that's said when you're on an interview.

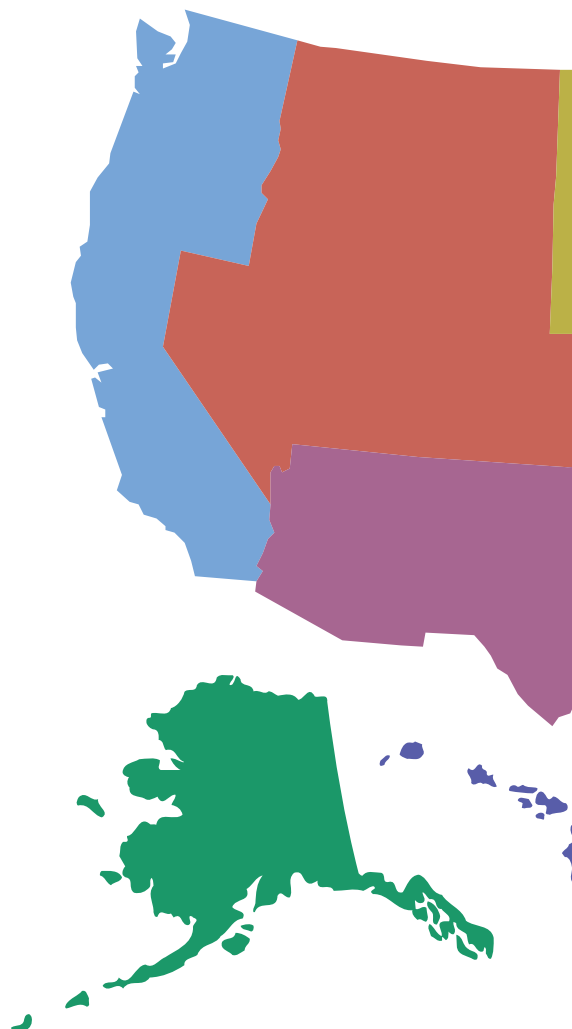
Ultimately, what you should do to further your career really does depend on your current skills and credentials as well as where it is that you'd like to see your career go. Whether it's a cushy government job with all the security in the world or a private contracting position working with bleeding-edge technologies on large scale applications with really high pay and a ton of risk, there is always something you could be doing to better the odds that five years from now you will be exactly where you want to be. I hope that this editorial has inspired and assisted some of our readers to get there. 

ColdFusion

For more information go to...

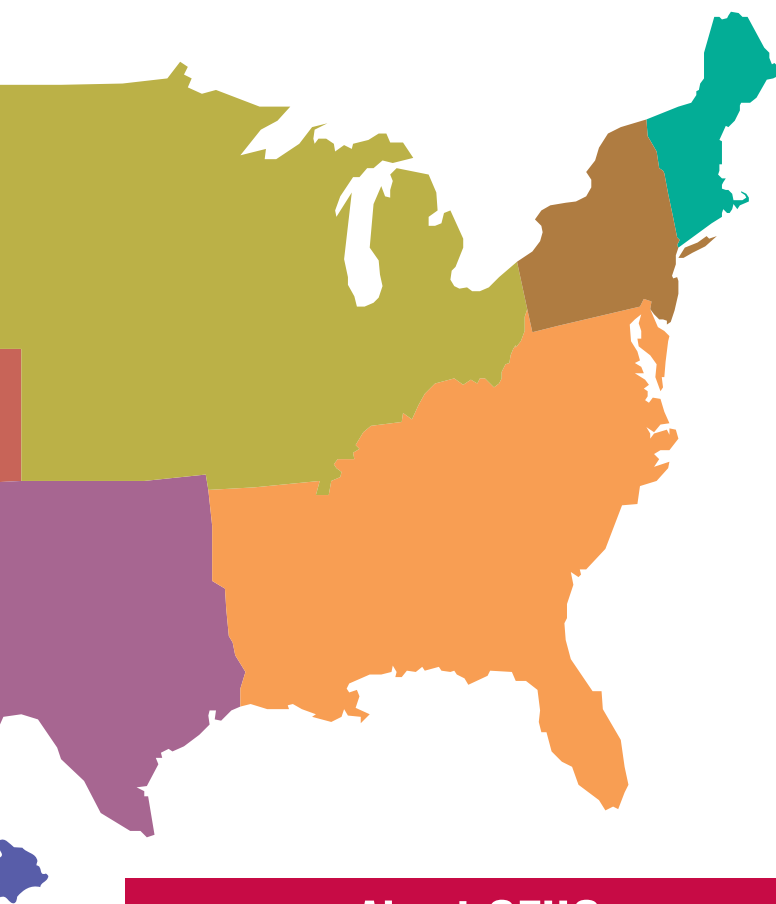
U.S.

Alabama Huntsville, AL CFUG www.nacflug.com	Louisiana Lafayette, LA MMUG http://www.acadianammug.org/	New York Albany, NY CFUG www.anycfug.org
Arizona Phoenix, AZ CFUG www.azcfug.com	Maryland California, MD CFUG http://www.smdcfug.org	New York New York, NY CFUG www.nycfug.org
California Bay Area CFUG www.bacflug.net	Maryland Maryland CFUG www.cfug-md.org	New York Syracuse, NY CFUG www.cfugcny.org
California Sacramento, CA CFUG http://www.saccfug.com/	Massachusetts Boston CFUG http://bostoncfug.org/	North Carolina Raleigh, NC CFUG http://tacfug.org/
California San Diego, CA CFUG www.sdcfug.org/	Massachusetts Online CFUG http://coldfusion.meetup.com/17/	Ohio Cleveland CFUG http://www.clevelandcfug.org
Colorado Denver CFUG http://www.denvercfug.org/	Michigan Detroit CFUG http://www.detcfug.org/	Oregon Portland, OR CFUG www.pdxcfug.org
Connecticut SW CT CFUG http://www.cfugitives.com/	Michigan Mid Michigan CFUG www.coldfusion.org/pages/index.cfm	Pennsylvania Central Penn CFUG www.centralpenncfug.org
Connecticut Hartford CFUG http://www.ctmug.com/	Minnesota Southeastern MN CFUG http://www.bittercoldfusion.com	Pennsylvania Philadelphia, PA CFUG http://www.phillycfug.org/
Delaware Wilmington CFUG http://www.bvccfug.org/	Minnesota Twin Cities CFUG www.colderfusion.com	Pennsylvania State College, PA CFUG www.mmug-sc.org/
Florida Jacksonville CFUG http://www.jaxcfusion.org/	Missouri Kansas City, MO CFUG www.kcfusion.org	Tennessee Nashville, TN CFUG http://www.ncfug.com
Florida South Florida CFUG www.cfug-sfl.org	Nebraska Omaha, NE CFUG www.necfug.com	Tennessee Memphis, TN CFUG http://mmug.mind-over-data.com
Georgia Atlanta, GA CFUG www.acfug.org	New Jersey Central New Jersey CFUG http://www.cjcfug.us	Texas Austin, TX CFUG http://cftexas.net/
Illinois Chicago CFUG http://www.cccfug.org	New Hampshire UNH CFUG http://unhce.unh.edu/blogs/mmug/	Texas Dallas, TX CFUG www.dfwcfug.org/
Indiana Indianapolis, IN CFUG www.hoosierfusion.com	New York Rochester, NY CFUG http://rcfug.org/	Texas Houston Area CFUG http://www.houcfug.org



User Groups

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



About CFUGs

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

INTERNATIONAL



Australia
ACT CFUG
<http://www.actcfug.com>

Australia
Queensland CFUG
<http://qld.cfug.org.au/>

Australia
Victoria CFUG
<http://www.cfcentral.com.au>

Australia
Western Australia CFUG
<http://www.cfugwa.com/>

Canada
Kingston, ON CFUG
www.kcfug.org

Canada
Toronto, ON CFUG
www.cfugtoronto.org

Germany
Central Europe CFUG
www.cfug.de

Italy
Italy CFUG
<http://www.cfmentor.com>

New Zealand
Auckland CFUG
<http://www.cfug.co.nz/>

Poland
Polish CFUG
<http://www.cfml.pl>

Scotland
Scottish CFUG
www.scottishcfug.com

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

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

Spain
Spanish CFUG
<http://www.cfugspain.org>

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

Switzerland
Swiss CFUG
<http://www.swisscfug.org>

Turkey
Turkey CFUG
www.cftr.net

United Kingdom
UK CFUG
www.ukcfug.org



Don't Talk to Strangers!

ColdFusion components and the Law of Demeter



By Kola Oyedeji

Don't talk to strangers" may sound like the advice your mother used to give

you as a child, but as is so often the case,

mother knows best. As we'll soon see,

"don't talk to strangers" or "only talk to

your friends" is advice that can apply to programming as well. In programming terms, this general concept is known as the Law of Demeter, and it's a great programming practice that facilitates a reduction in coupling between objects in object-oriented applications.

Before you turn to the next article and dismiss this as applicable to OO programming but having no relevance to your procedural Fusebox application, read on as you'll see that the benefits of the Law of Demeter can easily be gained in any application that uses ColdFusion components (CFCs). By applying the Law of Demeter, you can reduce coupling between your CFCs and make your application more flexible and easy to maintain.

First let's investigate the concept of coupling. Coupling is the degree to which one object is dependent upon another or, in the case of ColdFusion, the degree to which one CFC is dependent upon another CFC. For example, if CFC A is used by CFCs B and C and the CFCs are tightly coupled, a change to CFC A will also likely impact CFCs B and C. Minimizing coupling between CFCs is an important goal as you develop your applications.

Have you ever worked on an application in which you have changed something that appears to be quite a small, isolated part of the system, but changes to this seemingly isolated part

result in the application grinding to a halt? Perhaps that's an extreme example, but you don't want to make a change to a ColdFusion component in an application and have it all come crashing down because of the huge amount of dependencies between that component and other components within the application.

Systems that have low coupling reduce the impact of changes, thus making them easier to maintain. We all like things to be easier to maintain, right?

With all this talk of coupling, you may be getting the impression that any coupling whatsoever is a bad thing to have in your application. A key point to remember, however, is that a small amount of coupling is necessary in order for your application to do anything. It is unnecessary coupling that we should strive to avoid. In addition, coupling to stable, well-designed components such as the components in the ColdFusion MX 7 Admin API is generally safe as these are unlikely to change significantly in a way that will break backward compatibility.

As you build CFCs and attempt to avoid unnecessary coupling, it's helpful to think of each CFC as exposing an API to other components in the system. By exposing a public API, each CFC can hide its inner workings from other components so that outside components need only know how to communicate with the CFC, but they don't need to know any of the details related to how the CFC performs its functions.

In ColdFusion, components can be coupled in one of the following ways:

- If CFC *X* calls a function on CFC *Y*, component *X* is said to be coupled to component *Y*.
- If CFC *X* contains a variable (either in a variables scope or locally inside a function) that is a reference to CFC *Y*, component *X* is said to be coupled to component *Y*.
- If CFC *X* extends CFC *Y*, component *X* is said to be coupled to component *Y*.
- If CFC *X* receives an instance of CFC *Y* as an argument to one of its functions or an instance of CFC *Y* is returned from a function called on another CFC, then...

Well, you're probably starting to get the picture. Recognizing coupling and avoiding unnecessary coupling is an important

“Businesses that ignore the potential of SOA will find themselves outpaced by rivals who improve their agility and transform themselves into new kinds of enterprises

— Yafim Natis, Gartner Analyst

3-DAY EVENT!

SOAWorld

Plus **2007**

Enterprise OpenSource

Conference & Expo 2007

TOPICS INCLUDE:

SOA Web Services

- > AJAX and SOA
- > Web 2.0
- > Universal SOA
- > Protecting Web Services
- > Troubleshooting SOA
- > Governance
- > Open-Source SOA
- > XBRL
- > Service Virtualization

Open Source

- > Open Source Business Models
- > Open Source ESB
- > OpenAjax Alliance
- > SaaS and Open Source
- > Spring, Hibernate and Eclipse
- > Seam
- > Open Source Penetration
- > Monetizing Open Source
- > Open Source Databases
- > AMQP
- > Open Source Middleware

June 25-27, 2007

Roosevelt Hotel / New York City

Register Online! www.SOAWorld2007.com

11th International
SOAWorld
CONFERENCE & EXPO

2007 is to many industry insiders shaping up to be a major inflection point in software development and deployment, with SOA, Web Services, Open Source, and AJAX all converging as cross-platform and cross-browser apps become the rule rather than the exception.

Accordingly the 11th International SOA Web Services Edge 2007 again seeks to offer comprehensive coverage and actionable insights to the developers, architects, IT managers, CXOs, analysts, VCs, and journalists who'll be assembling as delegates and VIP guests in The Roosevelt Hotel in downtown Manhattan, June 25-27, 2007

Co-located with the 2nd Annual Enterprise Open Source Conference & Expo, the event will deliver the #1 i-technology educational and networking opportunity of the year. These two conference programs between them will present a comprehensive view of all the development and management aspects of integrating a SOA strategy and an Open Source philosophy into your enterprise. Our organizing principle is that delegates will go away from the intense two-day program replete with why-to and how-to knowledge delivered first-hand by industry experts.

**Visit soaeosconference.sys-con.com for the most up-to-the-minute information including...
Keynotes, Sessions, Speakers, Sponsors, Exhibitors, Schedule, etc.**

2nd Annual
ENTERPRISE > 2007
OPENSOURCE
CONFERENCE+EXPO

SOAEOSCONFERENCE.SYS-CON.COM

REGISTER ONLINE TODAY

SAVE \$200!

(HURRY FOR EARLY-BIRD DISCOUNT)

BROUGHT TO YOU BY:



» **SOA World Magazine**
focuses on the business and technology of Service-Oriented Architectures and Web Services. It targets enterprise application development and management, in all its aspects.



» **Enterprise Open Source Magazine**
EOS is the world's leading publication showcasing every aspect of profitable Open Source solutions in business and consumer contexts.

SYS-CON EVENTS For more great events visit www.EVENTS.SYS-CON.COM

Exhibit and Sponsorship Info:

Call 201-802-3020 or email events@sys-con.com

skill that you'll want to develop as you build your applications.

The Law of Demeter

The Law of Demeter was first defined by Ian Holland while working on The Demeter project at Northeastern University in 1987. In brief, the Law of Demeter states that an object should only call methods that:

- Belong to itself (i.e., its own methods)
- Are on objects passed in as a parameter
- Are on objects created by the object

A full summary of the Law of Demeter can be found at <http://www.cmcrossroads.com/bradapp/docs/demeter-intro.html>. Translating this to ColdFusion, this means your component should only call its own functions, functions of components it contains, or functions of components passed in to its own functions. An easier way to remember this is basically a CFC should not call functions on components that are returned from other CFCs. For example, the following violates the Law of Demeter:

```
<cfset myPostalCode = person.getAddress().getPostCode() />
```

In the following example, we have a Person CFC that contains an address CFC held internally in the variables scope with a function named `getPostCode`, and `getPostCode` returns a postal code. To retrieve the person's postal code we may write something along the lines of this:

```
<cfset me = createObject("component","person") />
<cfset postcode = me.getAddress().getPostCode() />
```

Or maybe the following, which is really just a variation of the code above:

```
<cfset me = createObject("component","person") />
<cfset myHouse = me.getAddress() />
<cfset postcode = myHouse.getPostCode() />
```

In these examples, the Person component "me" is returning the address component on which we then call the `getPostCode` function. The problem with this is that we have now coupled this code to two components. Say, for example, the address is now stored internally in a structure, not a component, and the `getAddress` function returns a structure instead of an Address object, we have to change this code. Similarly, if the address component method changes, perhaps the name of the `getPostCode` method is changed or the `getPostCode` method is removed completely, again we have to change this code.

Of course, changes to your public API will impact the code that uses it, but taking the time to think through your public API and how functionality is exposed will pay dividends down the road. Encapsulating the internals of your components prevents other code from being written based on these internals. This frees you to change the implementation of such components without breaking code that uses your component (provided the arguments and return values are the same). This also is the reason it's often recommended to not expose variables in the THIS scope.

Applying the Law of Demeter, instead of directly referencing

the address component returned from the Person CFC, we add the following function to the Person component:

```
<cffunction name="getPostCode" returns="string" output="false" >
    <cfreturn variables.personsAddressCFC.getPostCode() />
</cffunction>
```

We then change our calling code to call this new method, which is really nothing more than a wrapper method delegating the call to the actual address component held in the variables scope of the component.

```
<cfset someGuy = createObject("component","Person") />
<cfset pc = someGuy.getPostCode() />
```

Now if we change how the address is stored in the person CFC, if we change it to a structure – and change the person `getPostCode` function:

```
<cffunction name="getPostCode" returns="string" output="false" >
    <cfreturn variables.personsAddressStruct.postCode />
</cffunction>
```

As you can see, following the Law of Demeter now shields the calling code from changes to the Address CFC. The address may be internally stored as a CFC, a struct, or a Java class, but

the type makes no difference to the code calling the `getPostCode` function. As long as a string is still returned from the function as expected, this shields calling code from the impact of changes. We have minimized the scope of what needs to be changed so that code that now simply needs to get the postcode is not impacted by changes to the address component.

As with any design decision, there is a trade-off. One disadvantage of strictly applying the Law of Demeter is that you can end up with CFCs that contain many functions that do nothing more than delegate calls to other CFCs.

I hope that through these examples you can see that the Law of Demeter can help reduce the amount of dependencies between components in your applications. This ultimately leads to easier application maintenance.

References

- <http://www.ccs.neu.edu/research/demeter/papers/law-of-demeter/oopsla88-law-of-demeter.pdf>
- <http://www.cmcrossroads.com/bradapp/docs/demeter-intro.html>

About the Author

Kola Oyedele is technical director of Vivid Lime, a London-based full services agency. Prior to that, he was a senior software developer for the Collinson Group, developing Enterprise Loyalty and Insurance Systems. Kola holds a BSc in Computer and Information systems. His blog is available at www.coolskool.blog-city.com.

kola@oyedele.net



Subscribe Today!

SAVE 16%

12 Issues for **\$89⁹⁹**

OFFER SUBJECT TO CHANGE WITHOUT NOTICE



- Exclusive feature articles
- Latest CFDJ product reviews
- Interviews with the hottest names in ColdFusion
- Code examples you can use in your applications
- CFDJ tips and techniques

That's a savings of \$29.99 off the annual newsstand rate. Visit our site at www.sys-con.com/coldfusion or call 1-800-303-5282 and subscribe today!

ColdFusion Developer's
Journal



Objects Everywhere

A solution to object-oriented spaghetti code



By Jeff Chastain

Many times object-oriented programming (OOP) is billed as the end-all

solution to cure the spaghetti code that can come from procedural style applications.

After all, you just have to stuff your logic

code into a component (big OOP buzzword – encapsulation), and now your code is instantly better, right? How hard is it to stick a `createObject` function call or a `<cfobject>` tag in when you need to access that bit of code? Can anybody look into the future and see a problem here?

Let's look at an example. Many object-oriented articles use cars as examples, so we'll stick with the trend. When we need to find out information about a given car, we simply instantiate the Car object as follows:

```
<cfset oCar = createObject('component', 'com.mydomain.Car').init() />
```

Nice and easy, right? However, our application is going to get a bit more complex. Our car is not going to do much without an engine and a transmission. An engine and a transmission are not a simple property though; these are also objects with their own properties. In order to build our model, we need a couple more lines of code.

```
<cfset oEngine = createObject('component',  
    'com.mydomain.Engine').init() />  
<cfset oTransmission = createObject('component',  
    'com.mydomain.Transmission').init() />  
<cfset oCar = createObject('component',  
    'com.mydomain.Car').init(oEngine, oTransmission) />
```

Still, not too bad. But what happens when the business model continues to grow? None of us have ever experienced anything called scope creep, right?

The Problem

Every time we need a new Car instance, we have to remember all of the different dependencies a Car object has (the engine, the transmission, etc.) and write all of the different `createObject` function calls (in the right order) while getting the component path for each one correct. But wait, at the moment, our Car object is



kind of dumb. We need to go to the database and load information about our car, right? We need to create some data access objects, each of which requires a data source object. The stack of objects we need just to create a car almost makes you think about going back to procedural programming!

Add in the difficulty of a component path changing or a new dependency being added to the car object and all of a sudden we have spaghetti code all over again. So much for OOP being the solution to all the world's (or at least software developments) problems!

Introducing Design Patterns

If OOP is not the solution, then Design Patterns must be, right? Well, maybe. Design Patterns are simply standard solutions to common issues that arise in software design. Many design patterns apply to object-oriented development, but not all of them. A design pattern is not a piece of code that you can simply plug into your application. It is more of a template that offers guidance on how to solve different problems that may arise. Some different examples of design patterns include:

- **Decorator Pattern:** Wrap an existing object with a new “decorator” and expand the functionality of the original object without making changes to the code.
- **Singleton Pattern:** Restrict implementation of certain objects to a single instance across the application.
- **Facade Pattern:** Provide a simplified interface to a larger collection of objects.
- **Observer Pattern:** Allow objects to interact with their environment without being tied to it.

The Factory Pattern

Today, we are going to look at the Factory Pattern, which will hopefully help clean up some of this mess we're in. In the real world, what does a factory do? It makes things, right? Think of the GM Bowling Green Assembly Plant. At one end you say “I want a new red Corvette,” and at the other end of the factory, out comes a new Corvette (a new instance). I don't know anything about creating engines or transmissions. All I did was ask the factory for a new Corvette. Somewhere within the factory is the knowledge of what exactly makes up a Corvette and where to find all of the parts. The factory takes care of building the car without my having any knowledge of these specific details. Wait, factories don't have to be this specialized. In fact, I can go to the GM Bowling Green Assembly Plant and ask for a new blue Cadillac XLR. Once again, my order goes in one end and out pops a car from the other end without my having any knowledge of exactly how the car was built. This one factory builds multiple kinds of cars, possibly using some of the same parts, but all of that implementation is hidden.

How does this apply to object-oriented development?

The Object Factory

Let's take the auto assembly plant and turn it into an object model. In the business case, I know that a Car is made up of a whole series of other objects – things like an engine, transmission, seats, and even the stereo system. I don't want to keep track of all of these dependencies every time I need a new car. The solution? An auto assembly plant or, in other words, an object factory. I need an object that I can ask to give me instances of

other objects. In this example, I would have a Car factory object that I could ask for different types of cars and the factory object would then give me the completed object without my code having to know anything about what makes up a Car.

Since we are developers, we like to see code. What would a factory object look like?

```
<cfcomponent>
    <cffunction name="getCar" access="public" returntype="com.myDomain.
Car">

        <cfargument name="make" type="string" required="true">

            <cfswitch expression="#arguments.make#">
                <cfcase value="corvette">
                    <!-- create corvette object dependent objects -->
                    <!-- create and return corvette car object -->
                </cfcase>
                <cfcase value="xlr">
                    <!-- create cadillac xlr object dependent objects -->
                    <!-- create and return cadillac xlr car object -->
                </cfcase>
            </cfswitch>

        </cffunction>
    </cfcomponent>
```

Now, anytime I need to create a new Corvette, I just ask my factory for one like this:

```
<cfset oCorvette = oCarFactory.getCar('corvette') />
```

This is much, much better. Now, we have one place for all instantiation code instead of having it spread throughout the application. If we ever need to change the path of a component or add a new dependency, we just have to make the change in the factory. However...

You had to say it, didn't you? If you look at the code for our car factory and look into the future a bit, you will see that really, what we have done is move the spaghetti code from our application into the factory object. The code within that factory object is going to get quite complex, as for each type of car we create, we have to know all of the dependencies that that type of car has. Some of these dependencies will more than likely be shared by multiple types of cars as well, which means if the instantiation of an engine changes, we'll have to find all of the instances of that code in our factory and update them. While this is better than nothing, there is still room for improvement.

The Smart Object Factory

Along the way, some very smart people have taken the concept of the object factory and expanded upon it. Here enters the term Dependency Injection. What if, instead of building a huge switch statement inside of my factory, my factory was smart enough to handle all of the dependencies itself? For example, what if I could define for the factory a blueprint that says when you build an instance of a Car object, it is dependent upon instances of an Engine object and a Transmission object. At this point, my only concern is the Car object, not how to build the Engine or Transmission object. Later on in the plans, I define

that when the factory builds an instance of an Engine, it is dependent upon an instance of a Cylinder object.

What have I done here? By separating the logic of building the Engine object away from the logic of building the Car object, I have encapsulated that logic. Now, if a Truck object also depends upon the Engine object, it can use the same set of blueprints that the Car object uses. This means if I have to change the way the Engine object is instantiated, I can do it in one place and let the factory handle putting all of the pieces together.

This sounds a whole lot more complicated than our little factory component. Never fear, somebody else has already done all of the heavy lifting for us. A framework has been developed called ColdSpring by Chris Scott and Dave Ross. The purpose of the ColdSpring framework is to “make the configuration and dependencies of CFCs easier to manage.” Sounds exactly like what we are looking for. Let’s look at some of the high-level help that ColdSpring can provide our application:

- ColdSpring moves all component paths outside of the application code.
- ColdSpring allows you to move or change implementations using single changes in XML.
- ColdSpring handles all object initialization.
- ColdSpring handles all object dependencies.

This almost sounds too good to be true. Let’s take a quick

look at the XML “blueprint” that we provide ColdSpring with in order for it to build our objects.

```
<bean id="Car" class="com.myDomain.Car">
  <constructor-arg name="Engine">
    <ref bean="Engine" />
  </constructor-arg>
  <constructor-arg name="Transmission">
    <ref bean="Transmission" />
  </constructor-arg>
</bean>

<bean id="Engine" class="com.myDomain.Engine" />
<bean id="Transmission" class="com.myDomain.Transmission" />
```

As you can see, we have specified a bean (object) definition for our Car object. The class (object path) is provided as well as a series of constructor arguments. This means, when ColdSpring creates a new instance of the Car object, it must also pass all of the listed arguments into that Car object’s constructor method. In this case, the arguments include an instance of the Engine bean and the Transmission bean. Later on in the file, we find the bean definitions for the Engine and Transmission beans.

By utilizing ColdSpring and its XML blueprint, we have significantly cut the amount of code necessary to create new objects. Even within our factory, we no longer have to know all of the details of the dependent objects when we are defining a given object.

Conclusion

While object-oriented programming offers a wide array of benefits, it is still possible to write bad OO code just as you can write bad procedural code. An object factory is one of many good practices that, when added to your bag of tricks, can help cure the object explosion that tends to happen in many OO applications.

If you would like to find out more about some of the topics we only had room to briefly touch on here, check out the following resources.

- Freeman, E.; Freeman, E.; Sierra, K.; and Bates, B. (2004). Design Patterns – Head First Design Patterns. O’Reilly: <http://www.oreilly.com/catalog/hfdesignpat/>
- Corfield, S. “Managing ColdFusion Components with Factories”: http://corfield.org/articles/cfobj_factories.pdf
- ColdSpring Framework (Chris Scott and Dave Ross) <http://www.coldspringframework.org/> 

About the Author

Jeff Chastain has been developing software application using object-oriented programming for over 12 years and has been developing Web applications in ColdFusion for over 8 years. He has experience in a variety of industries, from Fortune 500 companies to his own consulting practice. Currently, Jeff is an applications architect and systems developer for Alagad, Inc., and contributes to the blog at <http://alagad.com>.

jchastain@alagad.com

CFDJ Advertiser Index

ADVERTISER	URL	PHONE	PAGE
Adobe	www.adobe.com/products/coldfusion/flex2		2,3
CFDynamics	www.cfdynamics.com	866-233-9626	4
Cfunited	www.cfunited.com		15
CFDJ	http://coldfusion.sys-con.com/	201-802-3000	31
EdgeWebHosting	edgewebhosting.net	1-866-334-3932	35
HostMySite	www.hostmysite.com	877-215-4678	11, 26
Hotbanana	hotbana.com/cfdj	866-296-1803	6
Intergral	www.fusiondebug.com		13
JavaOne	javaone.sun.com/javaone/sf		19
Paperthin	www.paperthin.com	1-800-940-3087	21
SOAWorld	www.soaworld2007.com	201-802-3020	29
Virtualization Con	www.events.sys-con.com	201-802-3020	17
WebApper	www.seefusion.com		25

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



**For the greatest hits
of the 70's, 80's and 90's
call your web host's
tech support.**

For answers call us at 1-866-EDGEWEB
3 3 4 3 9 3 2

When calling your web host for support you want answers, not an annoying song stuck in your head from spending all day on hold. At Edgewebhosting.net, we'll answer your call in two rings or less. There's no annoying on-hold music, no recorded messages or confusing menu merry-go-rounds. And when you call, one of our qualified experts will have the answers you're looking for. Not that you'll need to call us often since our self-healing servers virtually eliminate the potential for problems and automatically resolve most CF, ASP, .NET, SQL, IIS and Linux problems in 60 seconds or less with no human interaction.

Sleep soundly, take a vacation, and be confident knowing your server will be housed in one of the most redundant self-owned datacenters in the world alongside some of the largest sites on the Internet today and kept online and operational by one of the most advanced teams of skilled Gurus, DBAs, Network and Systems Engineers.

By the Numbers:

- 2 Rings or less, live support
- 100% Guarantee
- 99.999% Uptime
- 2.6 GBPS Redundant Internet Fiber Connectivity
- 1st Tier Carrier Neutral Facility
- 24 x 7 Emergency support
- 24 Hour Backup
- Type IV Redundant Datacenter



**For a new kind of easy listening,
talk to EdgeWebHosting.net**

<http://edgewebhosting.net>



2003 - 2006

◦ Shared Hosting ◦ Managed Dedicated Servers ◦ Managed Colocation ◦ Semi-Private Servers
◦ ColdFusion ◦ BlueDragon ◦ ASP ◦ .NET ◦ .Linux ◦ .Java ◦ SQL Server ◦ .MySQL ◦ Self-Healing Servers

Other companies in this magazine spent a lot of time on pretty ads. As you can see, we did not. We spent our time hiring the best people and training them to deliver outstanding support for your website. We spent our time building a state of the art datacenter and staffing it with people who care about your website like it's their own. Compassion, respect, credibility, ownership, reliability, "never say no," and exceed expectations are words that describe our service philosophy. From the first time you interact with us, you'll see what a difference it really makes. And you'll also forgive us for not having a pretty ad.



WEB HOSTING • MANAGED DEDICATED SERVERS • COLOCATION • VPS • ECOMMERCE • BLOGGING • EMAIL