

COLD FUSION Developer's Journal

ColdFusionJournal.com

February 2006 Volume:8 Issue:2

PAGE 37 & 49

REAL-WORLD
AJAX
ONE DAY SEMINAR



March 13, 2006
New York, NY

April 24, 2006
San Jose, CA

www.ajaxseminar.com

SOA 10th International
WebServices
Edge
conference+expo



2006 ENTERPRISE
OPEN SOURCE
CONFERENCE+EXPO

June 5-6, 2006
New York, NY

soaconference.sys-con.com

CUSTOM ERROR HANDLING USING AJAX

ColdFusion
and AJAX 10

Mixins 18

Introducing
objectBreeze 24

Completing The Real
Estate Sample Application 34

14

Presorted
Standard
US Postage
PAID
St. Croix Press

PLUS...

2005 Readers' Choice Awards	32
ColdFusion User Groups	44
Easy Flash Dashboards	46



One little box. A whole lot of power.

Put it to work for you.
The shortest distance between you
and powerful web applications.

Full speed ahead. The release of Macromedia ColdFusion MX 7 is changing the whole game. This groundbreaking release introduces new features to help address your daily development challenges. These features include:



› **Structured business reporting? Check. Printable web content? Check.**

ColdFusion MX 7 provides a structured business reporting solution that will have you creating detailed business reports for anyone who needs them. You can also dynamically transform any web content into high-quality, printable documents with ease. The days of needing a third-party application to generate dynamic reports are going, going, gone.



› **Make rapid J2EE development a reality.**

So, you're heavily invested in J2EE but would love to complete projects in less time? ColdFusion MX 7 is your answer: It delivers RAD productivity on top of the power and scalability of J2EE and deploys onto standard J2EE server environments.



› **New mobile applications are a go.**

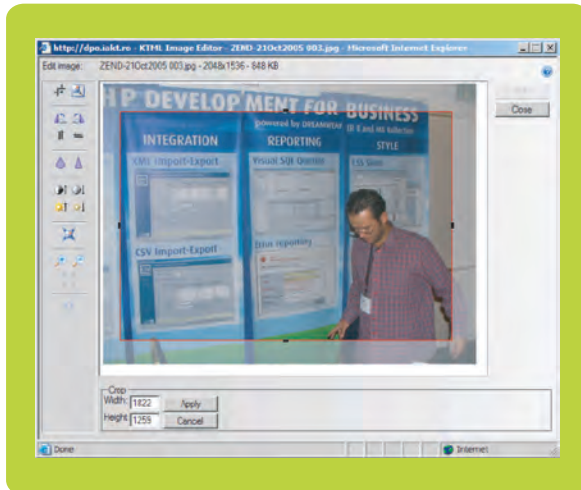
Innovative new features enable ColdFusion MX 7 applications to reach beyond the web. So you can rapidly create new applications, or extend existing ones, to connect with mobile phones and IM clients using a variety of protocols. The new world of mobile communications is exciting, and this your invitation to the party.

To learn more or take ColdFusion MX 7 for a test drive, go to:
macromedia.com/go/cfmx7_demo

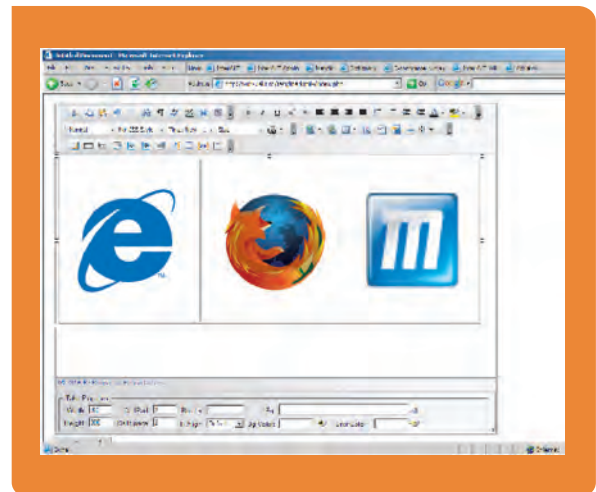
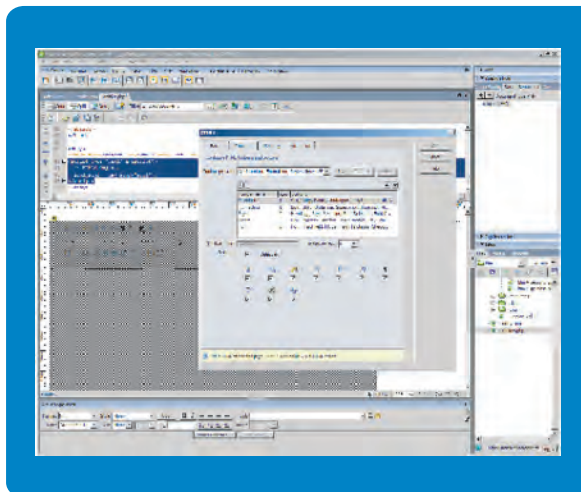
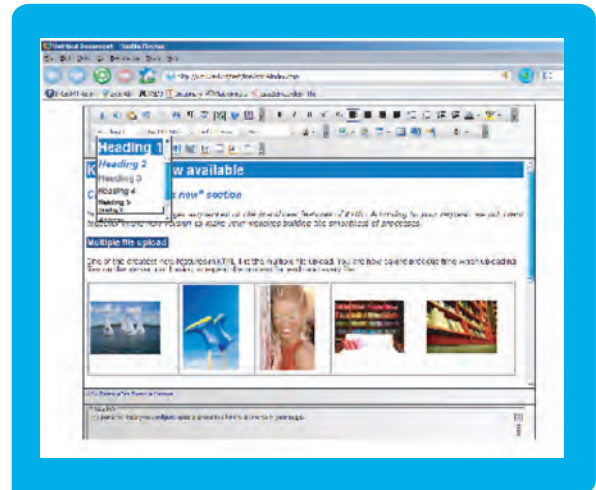
KTML4 Word editing in browser

Along with the most affordable **UNLIMITED** licence

Revolutionary Image Editor



Word-like visual CSS styles



Fast Dreamweaver integration

- Instant paste from Word
- Incredible speed
- Easy to use Word-like toolbars
- Improved CSS authoring
- Remote File Explorer
- XHTML 1.1 compliant

Wide browser compatibility

- Multiple file upload at once
- HTML Table Editor
- Support for multimedia (Flash, Avi)
- Documents management (.doc, .pdf)
- Page templates
- WAI compliant

Please visit www.interaktonline.com/ktml4/ for details

work smart

Interakt



Dedicated Server Packages Starting at \$189/mo.

All dedicated servers include:

- ▶ FREE STATS SOFTWARE!
- ▶ No long term commitments!
- ▶ FREE SQL server access!
- ▶ FREE MAIL SOFTWARE!
- ▶ Fair and simple pricing!
- ▶ Optional server maintenance!

As one of the premier ColdFusion hosting community leaders, CFDynamics is constantly looking for ways to provide **better service** to ensure your satisfaction. Not only do we offer the **finest in shared hosting plans**, but we now offer the **finest in 100% dedicated server plans**! Now you can afford the freedom of having your own dedicated server!

When your needs have outgrown shared hosting look to CFDynamics for **total freedom**. With dedicated server packages they're not offering an oxymoron; "virtually private" or "virtually dedicated" is **NEITHER** private nor dedicated. CFDynamics offers a solution that is **100% completely dedicated**. They don't play games with the fake stuff; CFDynamics only offers the real deal. Real Service. Real Satisfaction. Real Value.

Real Freedom.



Visit us online or call to order!



macromedia
ALLIANCE PARTNER

Paper|Thin Partner



BlueDragon Certified
V.I.P. Hosting Partner

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

technical editor

Raymond Camden raymond@sys-con.com

editor

Nancy Valentine nancy@sys-con.com

associate editor

Seta Papazian seta@sys-con.com

research editor

Bahadır Karuv, PhD bahadir@sys-con.com

production

production consultant

Jim Morgan jim@sys-con.com

lead designer

Abraham Addo abraham@sys-con.com

art director

Alex Botero alex@sys-con.com

associate art directors

Louis F. Cuffari louis@sys-con.com
 Tami Beatty tami@sys-con.com
 Andrea Boden andrea@sys-con.com

contributors to this issue

Ryan Anklam, Laura Arguello, Tim Burton, Nahuel Foronda,
 Hal Helms, Simon Horwith, Jeffrey Houser, Nicholas Tunney

editorial offices

SYS-CON MEDIA

135 Chestnut Ridge Rd., Montvale, NJ 07645
 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
 135 Chestnut Ridge Rd., Montvale, NJ 07645

©copyright

Copyright © 2006 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@edithrom.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.

What We Need Are Better Programmers



By Simon Horwith

It's no secret that I've been outspoken about not liking frameworks for quite some time now. The truth is, I believe that frameworks have a lot to offer. The most significant benefit that organizations stand to gain from using frameworks is a standardized way

to code and an environment that is generally more conducive to allowing multiple developers to work on a project at the same time. If frameworks help to standardize how things are done and make it easier for many developers to work on a project, why have I been vocal about not liking them? Am I just trying to create controversy?

The truth is that by and large, yes, I am trying to cause controversy...but only because I want to make people think – a statement that I make for a very good reason. I don't like frameworks because I've met a lot of developers over the years who have maybe one or two years of experience building applications with a framework but who don't have a good grasp of the CFML programming fundamentals you'd expect from someone with a few years' experience. I've met many developers who aren't capable of developing anything outside the context of the framework they're used to. In these scenarios, the framework is a crutch.

I view poor developers as a symptom of frameworks. Is this the fault of frameworks? No, not at all. There's nothing about a framework that says you can't use it and still be a terrific programmer. It's the fault of the developers, for sure, and it is for that reason that I don't truly dislike frameworks. I don't think any one framework should be used to build every application, and I personally prefer to develop without them, but when it's

appropriate I see nothing wrong with using them. If developers are really good at what they do, they should feel comfortable and be competent working without a framework and/or bouncing in and out of several frameworks. This is unfortunately not the case for the majority, though not all, of the people who use frameworks. It is especially true for developers who began their programming career working on projects that are built with frameworks and have never had the need to build an application without one.

You might be surprised to hear that I have a similar dislike of object-oriented programming and an even stronger distaste for design patterns. That's an extremely controversial statement to make and I'm sure that if my commentary on frameworks didn't upset or confuse the majority of our readers, talking poorly about OOP and design patterns will. Now that I've managed to get everyone worked up, let's talk about it.

What could I not like about object-oriented programming? Similar to frameworks, I actually do like OOP, but I don't

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. He has been using ColdFusion since version 1.5 and specializes in ColdFusion application architecture, including architecting applications that integrate with Java, Flash, Flex, and a myriad of other technologies. In addition to presenting at CFUGs and conferences around the world, he has also been a contributing author of several books and technical papers. You can read his blog at www.horwith.com. simon@horwith.com

Complex JAVA J2EE Hosting made easy.

WebAppCabaretsm

<http://www.webappcabaret.com/jdj.jsp>
1.866.256.7973



JAVA J2EE-Ready Managed Dedicated Hosting Plans:

Xeon III

**SAMEDAY
SETUP**

Dual 3.2 GHz Xeons
4GB RAM
Dual 73GB SCSI
1U Server
Firewall
Linux
Monitoring
NGASI Manager

\$399
monthly

Pentium 4 I

**SAMEDAY
SETUP**

**FREE
SETUP**

2.4 GHz P4
2GB RAM
Dual 80GB ATA
1U Server
Firewall
Linux
Monitoring
NGASI Manager

\$199
monthly
**2nd month
FREE**

4Balance I

1 Database Server
and 2 Application
Servers connected
to 1
dedicated load
balancing device.
Dual Xeons.
High-Availability.

\$1724
monthly

NEW! Geronimo 1.0 Hosting . Virtual Private Servers(VPS) starting at \$69/month

At **WebAppCabaret** we specialize in **JAVA J2EE Hosting**, featuring **managed dedicated servers** preloaded with most open source JAVA technologies.
PRELOADED WITH:

JDK1.4 . JDK1.5 . Tomcat . Geronimo . JBoss . Struts . ANT . Spring . Hibernate
Apache . MySQL . PostgreSQL . Portals . CRM . CMS . Blogs . Frameworks
All easily manage via a web based control panel.

Details:

- All Servers installed with the latest Enterprise Linux
- Firewall Protection
- Up to 60 GB daily on site backup included at no extra charge per server.
- Database on site backup every 2 hours
- Daily off site database backup
- A spare server is always available in case one of the server goes down
- Intrusion detection.
- 24x7 Server and application monitoring with automatic self healing
- The Latest Bug fixes and Security updates.
- Tier 1 Data Center. 100% Network Uptime Guarantee
- Guaranteed Reliability backed by industry-leading Service Level Agreements

Log on now at <http://www.webappcabaret.com/jdj.jsp> or call today at
1.866.256.7973

WebAppCabaretsm

JAVA J2EE Hosting

Prices, plans, and terms subject to change without notice. Please log on to our website for the latest price and terms. Copyright © 1999-2006 WebAppShowcase • All rights reserved • Various trademarks held by their respective owners.



president & ceo

Fuat Kircaali fuat@sys-con.com

vp, business development

Grisha Davida grisha@sys-con.com

group publisher

Jeremy Geelan jeremy@sys-con.com

advertising

senior vp, sales & marketing

Carmen Gonzalez carmen@sys-con.com

vp, sales & marketing

Miles Silverman miles@sys-con.com

advertising director

Robyn Forma robyn@sys-con.com

advertising manager

Megan Mussa megan@sys-con.com

associate sales manager

Kerry Mealia kerry@sys-con.com

sys-con events

president, events

Grisha Davida grisha@sys-con.com

national sales manager

Jim Hanchrow jimh@sys-con.com

customer relations

circulation service coordinator

Edna Earle Russell edna@sys-con.com

manager, jdj store

Brunilda Staropoli bruni@sys-con.com

sys-con.com

vp, information systems

Robert Diamond robert@sys-con.com

web designers

Stephen Kilmurray stephen@sys-con.com

Wayne Uffelman wayne@sys-con.com

online editor

Roger Strukhoff roger@sys-con.com

accounting

financial analyst

Joan LaRose joan@sys-con.com

accounts payable

Betty White betty@sys-con.com

accounts receivable

Gail Naples gailn@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

like what it's done to ColdFusion developers. This is truer for design patterns, which I view as the buzzword of the day right now. The reasons behind these statements are simple to explain, but very difficult to do anything about. I spend a lot of my "free time" reading about object theory and various thoughts and approaches to software architecture. The vast majority of the reading is not specific to ColdFusion but is certainly applicable. Likewise, my statements about OOP and design patterns are not specific to CF but apply to developers in all environments. An overwhelming number of programmers, not just CF but Java, .NET, C++, etc., talk about object-oriented programming without really understanding it at all. The roots of OOP are grounded in ideas, radical at the time they came out, about how to *think* about software development. Most of the books, courses, and, therefore, developers today have completely missed the point when it comes to OOP. These misconceptions and misapplied ideas are magnified greatly when developers begin throwing around talk about design patterns, which is why I said that I don't like them. The truth is, I think design patterns are an excellent learning tool and I am an avid supporter of OOP... but only when they're approached the right way.

My definition of "the right way" is for developers to learn "Object Think." Object Think is an approach to thinking about the objects that make up an application and is used for object modeling solutions. The original OOP concepts were based on this line of thought but it was quickly lost among developers, as they tend to use sequential thought when formulating solutions. Sequential thought comes more naturally to most people with a background in traditional programming and, to be honest, tends to be a more natural line of thought when formulating machine solutions, as programmers do. I have spent the better part of my free time for the past five or six years studying the writings and interpretations of the architect Christopher Alexander, software architecture pioneer Dr. David Parnas, Kent Beck (best known as the founding father of XP), and a handful of other early thinkers in the realm of object thought

and behavioral programming. To apply these philosophies and thoughts to ColdFusion development requires changing the way we plan and execute solutions. At the heart of this change is the need to be more object-centric, achievable via object-personification, in our approach to planning applications.

Describing a philosophy of how to model applications is obviously beyond the scope of this editorial. It's beyond the scope of an article and, in all fairness, is even difficult to achieve within the limitations of a book. My Sensible Assembly Methodology is based on these teachings, taken from Streamlined Object Modeling, Object Think, other thoughts and writings, and my architecture and development experience in the real world. Due to an extremely hectic schedule, I should have some documentation on that by the end of March at the latest, with a book to follow. My advocating a methodology (SAM) as opposed to a framework is driven by a personal belief that by doing so, I can

offer developers something more valuable and transferable – a better way to think about development.

What is important to me, for the good of the community, is not whether developers choose to use one framework or another or choose to use my methodology or another; it is that we not only accept that the way we solve problems and develop software could

be better, but examine every alternative approach in our attempts to find better ways to work. This idea has become more important with the recent buzz about Web 2.0 and next-generation Web applications. Simply throwing a Flex or AJAX front end on our application isn't good enough – in order to develop a true next-generation interface we have to change the way we think about our users and the experiences they should have when they interact with what we create. This is something that the XD (extreme design) group at Macromedia has been talking about for several years. As applied to software architecture, these ideas have been discussed for several decades. My hope is that now, with the vast amount of available information and the mature tools and technologies available to developers, the ColdFusion community can begin to put these ideas into practice.



TABLE OF CONTENT



Custom Error Handling Using AJAX

By Ryan Anklam...14



Introducing objectBreeze

By Nicholas Tunney...24



Completing The Real Estate Sample Application

By Laura Arguello & Nahuel Foronda...34

editorial

What we need is better Programmers

Changing our approach to planning application

By Simon Horwith

5

cf 101

ColdFusion and AJAX

A primer

By Jeffry Houser

10

foundations

Mixins

Writing software that's more flexible and maintainable

By Hal Helms

18

readers' choice award

2005 CFDJ Readers' Choice Awards

Winners and runners-up in 12 award categories

32

cfugs

ColdFusion User Groups

44

flash

Easy Flash Dashboards

Using CFMX-delivered dynamic data

By Tim Burton

46



**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.



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

<http://edgewebhosting.net>

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



2003 - 2006

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

ColdFusion and AJAX

A primer



By Jeffrey Houser

One of the most annoying things about the Web page paradigm is that you have

to reload the page whenever you want to

do something on the server side. If you

want to do a site search, it's sprinkled

across two pages.

On the first page, the user will enter a search term, then clicks a submit button to get to the second page where the search results are returned. If the user wants to filter or sort the search results it's usually refreshed again and each result is redisplayed in its filtered state.

Lately there's been a movement to help improve the Web by eliminating these constant page refreshes. Macromedia's Rich Internet Application (RIA) push with Flash and Flex has been at the forefront of this effort, however Flex' entry price has made it prohibitive for many ColdFusion Developers. That's where AJAX comes in. It's an alternate way to create a RIA and relies on technologies built into most browsers, making it completely free.

Dissecting AJAX

AJAX stands for Asynchronous JavaScript and XML. It's not a single technology, but rather a combination of different technologies that are used to create a Rich Internet Application feel. You've probably seen examples of this kind of application. Google uses AJAX a lot on such services as maps.google.com. After bringing up a location, you can click and drag the map, zoom in or out, or even bring up directions to and from the location. All this is done without any screen refreshes. You have to admit that the interface is pretty sweet. (Note: Yahoo has a similar mapping application, built entirely in Flex at <http://maps.yahoo.com/beta>.)

Since AJAX is a combination of technologies you're going to want to know what you're getting into before going any farther, right? I thought so. Here's the list:

- **XHTML:** To start with you need a display language. Most resources on AJAX say you have to use XHTML, but regular HTML will work too.

- **XMLHttpRequest:** The XMLHttpRequest object lets you retrieve data from the server without refreshing the page. It's a JavaScript object implemented in most major browsers. As the XMLHttpRequest object is being called, the user can do other things on the page. This is where the "asynchronous" part of the acronym comes in.
- **iFrames:** iFrames are in-line frames and are sometimes used as an alternative to the XMLHttpRequest object. A regular frame will divide the browser window into multiple separate pages, but an in-line frame embeds one page inside another. I won't get into any iFrames details in this article.
- **XML:** You all know what XML is, right? As a refresher, it's a way to describe data. If you need more details, read the article in this column from August 05 at <http://coldfusion.sys-con.com/read/117667.htm>.
- **Document Object Model:** The document object model defines your HTML page through a programmer API.
- **JavaScript:** I'm sure you all know what JavaScript is too. It's the most common language used for providing client-side functionality to HTML pages.

How do all these different technologies work together using AJAX? Well, it generally works something like this:

1. First, the user comes to your page and the page loads. Something (depending on what your page does) is loaded and displayed to the user.
2. The user does something such as clicking a link. Instead of reloading the page (as would happen in a normal Web app), that link fires off a JavaScript function.
3. The JavaScript function uses XMLHttpRequest object to retrieve XML data from the server.
4. Then the JavaScript parses the XML Data and through the Document Object Model will change the page without a refresh.

So the user gets to see more data quicker without the annoying page refreshes. That's fantastic, right? Why isn't everyone using this? Well, as with all things, AJAX isn't perfect. There are drawbacks:

- **Development Time:** Working with AJAX will bring you back to the old days of Web development, especially if you want any kind of cross-compatibility. There are different syntaxes for creating the XMLHttpRequest object in IE vs Netscape/Firefox, which can lead to extended development times.
- **User Experience:** When used properly, an AJAX can enhance the user experience. However, when used improperly it can



WEB ALL-STAR seeks an Integrated Software Suite that's up for anything. Understand my need to create cool graphics one day, while tweaking CSS the next. Should be intuitive, multi-platform and not phased by my turbo-charged pace. High-maintenance is a no-no.



Different people. Different needs. One suite solution.

With the latest versions of Macromedia Dreamweaver®, Flash® Professional, Fireworks®, Contribute™, and FlashPaper™, the new Studio 8 is quite a catch. To meet Studio 8 and find all the tools you need to design, develop and maintain online experiences, visit www.macromedia.com/go/studio8_mxdj

macromedia®
STUDIO 8

make it horrendous. Since the page isn't reloading, what happens when the user hits the back button? He'll leave the application altogether. I bet he expected to go to the app's "previous state." What happens when the user bookmarks a page? He'll go the "initial" state of the app. There are work-arounds, of course, but they're not trivial to implement.

- **Searches:** The single-page nature of the app makes indexing by search engines a tetchy proposition at best.
- **Response Times:** The very thing that works for you (no page reloads) can also work against you. What happens if you're retrieving a lot of data from the server? And the user is given no indication that something is happening.
- **JavaScript:** JavaScript must be enabled for an AJAX app to work. If JavaScript is disabled, your user isn't going to have a lot of fun with an AJAX app.
- **Accessibility:** AJAX is, most likely, not going to meet accessibility guidelines. If this is a goal of your project you're going to have to provide a fallback option. That means developing two paths to the same goal.

There are a few real simple examples of AJAX that you've probably used or seen that existed long before the term AJAX was conceived. Have you ever rolled your mouse over a graphic navigation button only to have that graphic change? JavaScript rollovers can be considered a precursor to AJAX. They do use JavaScript and they do change the state of the page. Have you turned on or off a DHTML layer on a page based on some selection by the user? This is another example of AJAX. If an application has a select box with an "other" option in it I'll often use DHTML to display an input text box if "other" is selected. If other isn't selected the input box is hidden. Both of these are simple examples of changing the page without a screen refresh. They don't go out to the server to get data, but they still fit the RIA paradigm.

Putting This All Together

Let me show you how this all comes together, so you can actually do something useful. The first step in the process is to initialize the XMLHttpRequest object. Unfortunately, there's not consistent way to do this across browsers:

```
<script language="javascript" type="text/javascript">
var request = false;
try {
    request = new XMLHttpRequest();
}
catch (trymicrosoft) {
    try {
        request = new ActiveXObject("Msxml2.XMLHTTP");
    }
    catch (othermicrosoft) {
        try {
            request = new ActiveXObject("Microsoft.
XMLHTTP");
        }
        catch (failed) {
            request = false;
        }
    }
}
```



```
}
</script>
```

The first line initializes the request variable, giving it a Boolean value of false. You can check against this later to make sure that your request object was properly initialized. The next code creates an instance of the XMLHttpRequest object using 'new XMLHttpRequest.' This works fine for most non-Microsoft browsers, but won't work in IE. If it doesn't initialize, then an error is thrown, which is where the first catch statement comes into play. It attempts to re-initialize using the method available in the current version of Internet Explorer: new ActiveXObject("Msxml2.XMLHTTP").

If that fails, it tries the version from older versions of Internet Explorer new ActiveXObject("Microsoft.XMLHTTP"). That's kind of complicated right? I borrowed this initialization code from the article <http://www-128.ibm.com/developerworks/Web/library/wa-ajaxintro2/?ca=dgr-lnxw07AJAX-Request> (which you should all read because it goes into a lot more details about error checking that are outside the scope of this article).

So, now the code has created the object, but what can you do with it? As you can probably see, it doesn't do much yet. You want to assign it a URL, load the data, and then process the data in some way. This function will do that:

```
function getRSS() {
    var url = "/htdocs/jeffhousercom/wwwroot/rss.cfm";
    request.open("GET", url, true);
    request.onreadystatechange = updatePage;
    request.send(null);
}
```

This is function to grab an RSS feed. The first line defines the URL. This is standard JavaScript for defining a variable. You should bear in mind that you can only request URLs from the current Web site. In this case, I'm grabbing a RSS feed from a local copy of my blog. The next line calls a method on the request object entitled open. Open prepares the request to be sent, but doesn't actual send it. There are five arguments to the open function (although we only use three in this example):

- **Request-type:** This specifies the request type, either get or post, for the URL that you want to send.
- **url:** The URL specifies the URL that you're getting (or posting) to.
- **Asynchronous:** The async value is a Boolean value that specifies whether this request is asynchronous or not. For AJAX use, you'll want to specify true. If set to false, the application will hold until this request is finished, which defeats the purpose of an RIA in the first place.
- **username:** The username specifies the username required for accessing the URL, if applicable.
- **password:** The password specifies the password required for accessing the URL, if applicable.

Okay, you've set up the request to be ready to send. The next line specifies the onreadystatechange property. This property speci-

fies what function to call once the request is complete. If you don't specify a method to be called then nothing will happen once the request is complete. The final line calls the send method. The send method is the one that will execute the request. Its parameter is the data that you need to send along with the request. In our example, we aren't sending parameters, so we pass a null value.

Once the request complete, the updatePage function will execute, so lets look at that next:

```
function updatePage() {  
    if (request.readyState == 4){  
        Form1.XMLValues.value = request.responseText;  
    }  
}
```

This function first checks the readyState of the XMLHttpRequest object. Exploring that is beyond the scope of this article, but there are a lot more details in some of the references provided at the end. For now, all you need to know is that if the readyState is 4, then the request is complete. This takes the resulting value from the server response and assigns it to a form textbox. The form is below:

```
<form name="Form1">  
    <textarea name="XMLValues" rows="20" cols="50"></textarea>  
</form>
```

The text box will contain the XML from the RSS feed. There's

another property associated with the XMLHttpRequest object called responseXML. This will contain the results as an XML Document object, which JavaScript can access natively. Unfortunately specific details are beyond the scope of this article, but I found this good resource to get you started on that: <http://www.peachpit.com/articles/article.asp?p=29307&seqNum=4&rl=1> .

Where To Go from Here

If you want to learn more about AJAX, there are some great resources you can read. The article by Jesse James Garrett that started it all is at <http://www.adaptivepath.com/publications/essays/archives/000385.php>. Garrett is credited with coining the term AJAX. You can follow an AJAX blog at <http://www.ajax-powered.net>, which talks about all things AJAX. Finally, there are a few people who are working on ways to integrate ColdFusion and AJAX together. One is the CFAJAX project <http://www.indiankey.com/cfajax>. Another is the AJAXCFC project at <http://www.robgonde.com/blog/projects/ajaxcfc>. 

About the Author

Jeff Houser has been working with computers for over 20 years. He owns a DotComIt, a web consulting company, manages the CT Macromedia User Group, and routinely speaks and writes about development issues. You can find out what he's up to by checking his Blog at www.jeffryhouser.com.

jeff@instantcoldfusion.com

Efficient Web Content Management

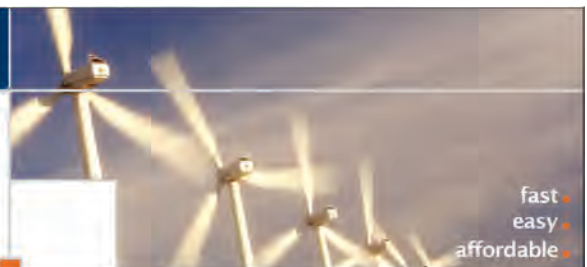
CommonSpot™ Content Server is the ColdFusion developer's leading choice for efficient Web content management.

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.

Custom Error Handling Using AJAX

Enhancing the interactive experience



By Ryan Anklam

AJAX has become an increasingly popular tool to develop RIAs. With AJAX, as with many new technologies, developers often overlook core application issues such as error handling.

While many current AJAX frameworks come

with ways to handle errors, the built-in error-handling methods might not be quite what you need, and it's possible that you might not even want to adopt a specific AJAX framework at all. So how do you handle errors in AJAX?

This article will guide you through one possible way to implement custom error handling in AJAX using many of the same techniques that you'll likely read about in other parts of this magazine. Because AJAX represents a big paradigm shift in the way users interact with Web applications, it's easy to leave your users confused when things don't quite work as they'd expected. To enhance the user experience, it's equally important to alert them when something goes exactly as planned and enhances the interactive experience. Consider for a moment a hypothetical AJAX application that updates employee information. Users will fill out fields and click

on a "Save" button to process the update. In a traditional Web application the user expects to wait a moment while the server updates the record then anticipates another page that displays a confirmation message. This is the typical request/return scenario that our Internet conditioning forces us to accept.

Now let's look at the same example using AJAX techniques. The end user still fills out form fields and clicks on the "Save" button but instead of seeing the confirmation message, nothing seems to happen. This can leave the user confused and unsure that his information was saved, yet with AJAX, the database update occurred exactly as expected. Here's how you can implement a messaging and error system in a simple employee information maintenance application that will alert a user as records are updated.

Process at a Glance

The process isn't much different from any typical AJAX request/response. A request is created, sent to the server, checked for error conditions, XML is sent back to your request page, and checked in the browser for a status message, which is displayed, if it exists.

Create an Area to Display Status Messages

We'll begin to create our code by creating our CSS format classes for our status messages. Let's create three styles for our application's potential conditions: error, success, and hidden, which correspond to the two cardinal states (error and success) of our query (hidden being used when no update is currently active):

```
.error{
```

```

font-family: Arial, Helvetica, sans-serif;
font-size: 10px;
font-weight: bold;
color: #FFFFFF;
background-color: #FF0000;
display: block;
}
.success {
font-family: Arial, Helvetica, sans-serif;
font-size: 10px;
font-weight: bold;
color: #FFFFFF;
background-color: #009900;
display: block;
}
.hidden {
display: none;
}
}

```

Once we have our styles set, we have to put them to use. We'll do this by creating an area in our application to display the status messages returned by the server. This display area can be any type of HTML container such as table, div, or span; we'll use a div. Once the container is created, we'll set the initial style to hidden, as follows:

```
<div id="message" name="message" class="hidden"></div>
```

Creating the ColdFusion Page To Process the Request.

At this point, you'll find that the code you've created doesn't do much — you have three CSS styles, one of which is called by your container div, but its class is set to not display on the rendered page. To create conditions where an error or success message might actually exist, we'll look at a ColdFusion template that updates an employee's database record. Since the focus of this article is on error handling we'll skip the details of updating the record and get right to the process of building the XML that returns our status message.

First let's look at an example of processing an update of one of our employee records. Since this is a situation where no data is being returned (because we're not using a SELECT statement), we really only need to deliver either a success or error message to our user.

To do this, we'll have to create a variable to hold the XML string then add the XML declaration to it:

```
<cfset xml = "<?xml version='1.0' encoding='UTF-8' standalone='yes'>">
```

Now that we've created our variable, "xml," we're going to want to do some simple data validation — in this case, to make sure that a valid department ID was passed into the template. If the department ID passed in is not valid, we're going to want to set the first node of the XML document to (<error>), add the error message, and close the error node (</error>). For our purposes, we'll assume that a "departmentID" value of 0 or of a non-numeric value constitutes an invalid condition. We're also going to include "try/catch" conditions to cover database errors and general failures:

```

<cff departmentId eq 0 OR NOT IsNumeric(departmentId)>
<cfset xml = xml & "<error>Invalid department specified.</error>">
<cfelse>
    <cftry>
    <cfcatch type="database">
        <cfset xml = "<?xml version='1.0' encoding='UTF-8'
standalone='yes'>" ?><error>There was a error communicating with the
server, please call the help desk at x555.</error>">
    </cfcatch>
    <cfcatch type="Any">
    <cfset xml = "<?xml version='1.0' encoding='UTF-8' standalone='yes'
?><error>There was a error processing your request. Please try again later
or call the help desk at x555.</error>">
    </cfcatch>
    </cftry>
</cff>

```

One thing to note here is that the entire XML string is overwritten in the catch statement. This eliminates a situation caused by an error being thrown in the middle of building the XML string. Specifically this condition sends an incomplete and unpredictable XML document back to the requesting page. Let's walk through an example of a successful and unsuccessful update of an employee record. The main screen of the employee update application is shown in Figure 1. For this example let's say that a phone extension can only be used once per employee in a company. When a user clicks on the "Update Employee" link the main screen will prepare the AJAX request and send it to a ColdFusion template. The template will then process the update and send a XML message back to the main screen.

If everything in the update goes okay the ColdFusion template will send the page a success XML message that would look something like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?><data><Success>Successfully updated employee record.</Success></data>
```

Now, let's say that the extension is in use. Our ColdFusion template will return a error XML message that would look something like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?><data><error>There was a error updating the employee, extension is in use by Desmond Mason. Please call the help desk at x555.</error></data>
```

Figure 1

Figure 2

To complete the application we have to have a way to handle the XML sent back to the requesting page.

Creating JavaScript to Handle the Errors

Let's take a look at what we've done so far: we've created our CSS style classes, we've created our display container, we have our query, our template to update the employee record, and created our error-handling logic. The next step is to read the response from the server, parse through the XML, and set the appropriate display condition. We'll start this step by creating a function to show error messages. This function will need two parameters: the message text and an indication of whether or not this is an error message.

We'll start writing this function by setting a variable with a value that will represent the name of the message area — this, again, is our display container. Now that we have our display container we have to set the class of the display container conditionally based on the error parameter. Finally, we can set the innerHTML property of message node equal to the message's value:

```
function ShowMessage(message, isError)
{
    messageArea = document.getElementsByName('message')[0];

    if(isError)
    {
        messageArea.className = 'error';
    }
    else
    {
        messageArea.className = 'message';
    }

    messageArea.style.display = 'block';
    messageArea.innerHTML = message;
}

if(response.childNodes[0].nodeName == 'error')
{
    ShowMessage(response.childNodes[0].firstChild.nodeValue, true);
}
```

After the ShowMessage function is written we have to write the logic to check the value of the first node returned. Since we called an update page we expect the first node to be named either “error” or “success” so we create code that will check to see if this is true:

```
if(response.childNodes[0].nodeName == 'error')
{
    ShowMessage(response.childNodes[0].firstChild.nodeValue,
true);
}
else
{
    ShowMessage(response.childNodes[0].firstChild.nodeValue,
false);
}
```

Let's look at our example of updating the employee record again. In the first case we got an XML message that looked like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?><data><Success>Successfully updated employee record.</Success></data>
```

In this case our script would call the ShowMessage function with the error Boolean value set to false. The resulting page would look something like Figure 2.

Now let's look at the same page if the server returned an error because the employee's extension is used by another employee. As mentioned above, the XML message returned from the ColdFusion template would look like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?><data><error>There was a error updating the employee, extension is in use by Desmond Mason. Please call the help desk at x555.</error></data>
```

In this case our script would call the ShowMessage function with the error Boolean value set to true. The resulting page would look something like Figure 3.

Coupling with Client-Side and Template Error Checking

To make the messaging system complete we have to integrate traditional ColdFusion error checking as well as client-side error checking.

Let's look at how to integrate traditional ColdFusion error handling into our messaging system first. We'll start by creating two session variables: one to hold a Boolean value to determine if an error has occurred and a second one that will be a string to hold the error message that we're going to return to the visitor. For this example I'll name them session.error and session.errorMessage. The next step is to wrap all your ColdFusion logic in the <cftry> and <cfcatch> statements. In your <cfcatch> statement you'll set the session.error to true and the session.

Figure 3

Figure 4

errorMessage equal to the cfcatch.Message.

```
try
{
    Department = CreateObject("Component","CFDJ.Components.Department");
    DepartmentArray = Department.GetAllDepartments(#session.dsn#);
}
catch(Any ex)
{
    session.error = true;
    session.errorMessage = ex.Message;
}
```

Finally we'll have to add some logic to control the visibility of our message area. If the session.error message is false the display area is created exactly as before. However, if an error is found we'll set the default style of the message area to "error" and display the error message:

```
<cfif NOT session.error>
    <span id="message" name="message" class="hidden"></span>
<cfelse>
    <span id="message" name="message" class="error">#session.errorMessage#</span>
    <cfset session.error = false>
</cfif>
```

Now let's say that the connection to the database was down when the user first entered the application. The function in Listing 1 is used to get a list of all the departments in the company.

If the database server is down the cfcatch will catch a database error and throw a new error message that will tell the user that the database server is down, it's up to the page that calls this function to trap the error and set the session.error and session.errorMessage variables:

```
<cfscript>
    try
    {
        Department = CreateObject("Component","CFDJ.Components.Department");
        DepartmentArray = Department.GetAllDepartments(#session.dsn#);
    }
    catch(Any ex)
    {
        session.error = true;
        session.errorMessage = ex.Message;
    }
</cfscript>
```


In this case the employee update page would look like Figure 4 when the user enters the application.

Next let's look at using JavaScript to do some simple client-side error handling. The first step is to check for invalid data. I'll assume that you've had some experience doing simple JavaScript error checking and skip right to displaying the error. For our example, we're going to have the JavaScript check to make sure that the user has selected the name of a user to edit

from the user drop-down. The first step would be to make sure that the selected index of a drop-down is not zero. Typically the next step would be to display some sort of message to the user with the Alert() function. In this case we'll simply call the ShowError() function that we wrote above:

```
if(departmentSelect.selectedIndex != 0)
{
    //do something
}
else
{
    ShowMessage('Please select an employee to update',true);
}
```

Conclusion

Adding a messaging system to an AJAX application isn't a very complicated, but it can be an extremely useful technique for keeping your end users informed. This application can be viewed online at <http://www.innovacreative.com/cfdj/ajax>. The source code can be downloaded from <http://www.innovacreative.com/cfdj/ajax/ajax.zip>. 

About the Author

Ryan Anklam is the Chief Information Officer at Innova Creative Media, Inc. His current focus is on using ColdFusion to develop large scale hosted applications. Ryan has been developing ColdFusion applications since 1996. In addition, he is also a Microsoft Certified Professional with demonstrated skills in C# and SQL Server.

Listing 1

```
<cffunction name="GetAllDepartments" access="public"
returntype="array" output="false">
    <cfargument name="dsn" required="true" type="string">
    <cftry>
        <cfquery name="GetAll" datasource="#arguments.dsn#">
            SELECT
            id
            FROM
            Department
        </cfquery>
        <cfscript>
            DepartmentArray = ArrayNew(1);
            //if no departments found, throw an error
            if(GetAll.RecordCount eq 0)
            {
                ThrowError('No departments found.');
```

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

Mixins

Writing software that's more flexible and maintainable



By Hal Helms

In the last two installments of “Foundations,” we looked at the issue of static v dynamic typing in ColdFusion and I argued that treating ColdFusion as a statically typed language led to disappointment and defeat. In this issue let’s

explore some of the possibilities available when we no longer treat ColdFusion as “Java Lite” but accept it on its own terms..

What Are “Mixins”?

In the late 1970s, a Lisp-based language called “Flavors”, originating at M.I.T., introduced the idea of adding functionality from one class to another. The term, *mixin*, was apparently inspired by some of the ice cream parlors around the M.I.T. campus that offered customers the option of mixing such goodies as Oreos and candy bars into their standard ice cream flavors.

I’ve been working with Ruby lately, which also offers mixins, and I began to wonder if, freed from the constraints of static typing, (1) it might be possible and, (2) we might benefit from implementing mixins in ColdFusion.

After some thinking, talking with colleagues, and experimenting, I came up with two methods of introducing mixins and I’ve given them the names *class-based mixins* and *object-based mixins*. Both are quite useful but work very differently.

Class-Based Mixins

In the November issue of *CFDJ*, I explained how Java interfaces are used to provide for multiple-type inheritance while avoiding the problems of code conflict that multiple inheritance languages must deal with.

Java interfaces are wonderful — for Java — but they do violate a sound fundamental of creating software, dubbed the DRY (Don’t Repeat Yourself) Principle. In an interview, Dave Thomas (of *The Pragmatic Programmer* fame) had this to say about DRY: “DRY says that every piece of system knowledge should have one authoritative, unambiguous representation.”

Java interfaces push the “authoritative, unambiguous representation” of methods down to the implementing classes. There are many situations that arise where multiple implementing classes of a single interface will mean multiple code implementations.

Let me offer some example Java code to illustrate:

```
public interface Teacher{
    public void giveTest(Test test);
}
```

Here we have a very simple interface, defining what it means to be a teacher: a teacher is someone who can give a test. What does it mean to give a test? For our purposes, let’s say that it entails the following:

1. Give out Test booklets
2. Answer any questions about the test
3. Monitor the students
4. Monitor the time
5. Collect the test booklets at the end of the allotted time

Initially I want to have a **Teacher** class that implements the **Teacher** interface. It might look something like this:

```
public class Teacher implements interfaces.Teacher{
    private String name = null;

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

    public void giveTest(Test test){
        giveOutTestBooklets(test);
        monitorStudents();
        monitorTime(test);
        collectTestBooklets();
    }
    // giveOutTestBooklets, monitor Students, monitorTime,
    // and collectTestBooklets methods below...
}
```

Now, I’ll implement the **Teacher** interface in another class — say, **StudentTeacher**:

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

```
public class StudentTeacher extends Student implements Teacher{

    public StudentTeacher(){
        // default constructor
    }

    public void giveTest(Test test){
        giveOutTestBooklets(test);
        monitorStudents();
        monitorTime(test);
        collectTestBooklets();
    }
    // giveOutTestBooklets, monitor Students, monitorTime,
    // and collectTestBooklets methods below...
}
```

That doesn't look like one representation to me: we have a lot of duplicated code. Fortunately, class-based mixins can help us. Let's tackle the same problem but with ColdFusion with class-based mixins.

Our first step is to create a single representation of a teacher. We'll save this as a ColdFusion template (not a CFC) called **Teacher**:

```
<cfset variables.instance.name = null />

<cffunction name="init" access="public" output="false">
    <cfargument name="name" required="true" />
    <cfset set('name', arguments.name) />
    <cfreturn this />
</cffunction>

<cffunction name="giveTest" access="public" output="false">
    <cfargument name="test" required="true" />
    <cfset giveOutTestBooklets(arguments.test) />
    <cfset monitorStudents() />
    <cfset monitorTime(arguments.test) />
    <cfset collectTestBooklets() />
</cffunction>

<!--- other methods below... --->
```

This is the code we're going to mix into classes. Our first class is **Teacher.cfc**:

```
<cfcomponent displayName="Teacher"
extends="BaseComponent">
    <cfinclude template="Teacher.cfm" />

    <cffunction name="init"
access="public" output="false">
        <cfargument name="test"
required="true" />
        <cfset set('test', arguments.
test) />
        <cfreturn this />
    </cffunction>
</cfcomponent>
```



The only thing needed is to include **Teacher.cfm** and write an **init** method — a sort of constructor for our CFC.

Our **StudentTeacher.cfc** is similar:

```
<cfcomponent displayName="StudentTeacher" extends="Student">
    <cfinclude template="Teacher.cfm" />

    <cffunction name="init" access="public" output="false">
        <cfargument name="average" required="true" />
        <cfargument name="name" required="true" />
        <cfargument name="test" required="false" default=null />
        <cfset super.init(arguments.average, arguments.name) />
        <cfset set('test', arguments.test) />
        <cfreturn this />
    </cffunction>
</cfcomponent>
```

Without extending or implementing anything, we have the one authoritative, unambiguous representation of what a teacher is.

That's not to say, though, that we can't extend classes. In fact, **StudentTeacher** extends **Student.cfc**:

```
<cfcomponent displayName="Student" extends="BaseComponent">
    <cfinclude template="Student.cfm" />

    <cffunction name="init" access="public" output="false">
        <cfargument name="average" required="true" />
        <cfargument name="name" required="true" />
        <cfset set('average', arguments.average) />
        <cfset set('name', arguments.name) />
        <cfreturn this />
    </cffunction>
</cfcomponent>
```

Student is similar to **Teacher** and includes the one authoritative, unambiguous representation of what a student is: **Student.cfm**:

```
<cfset variables.instance.average = 0 />
<cfset variables.instance.name = null />

<cffunction name="study" access="public" output="false">
    <cfreturn "Yes, yes, I'm studying..." />
</cffunction>
```

```
<cffunction name="takeTest" access="public"
output="false">
    <cfargument name="test" required="true" />
    <cfif arguments.test.get('percentOfGrade')
LT 10 AND get('average') GTE 100>
        <cfreturn skipTest() />
    <cfelse>
        <cfreturn workHardOnTest(arguments.
test) />
    </cfif>
</cffunction>
```

```
<cffunction name="skipTest" access="private" output="false">
  <cftrace type="Information" text="I think I'll just skip this test.
I've already got a #get('average')#" />
</cffunction>
```

```
<cffunction name="workHardOnTest" access="private" output="false">
  <cfargument name="test" required="true" />
  <cftrace type="Information" text="I'm taking the test: #arguments.
test.get('name')#" />
</cffunction>
```

We should make sure all this works:

```
<cfset anne = CreateObject('component', 'Student').init(94, 'Anne Baker')
/>
<cfset bob = CreateObject('component', 'Student').init(82, 'Bob Carter')
/>
```

```
<cfset carla = CreateObject('component', 'StudentTeacher').init(100,
'Carla Davis') />
```

```
<cfset test = CreateObject('component', 'Test').init('00 Development with
ColdFusion', 8) />
```

```
<!-- have the Teacher do its thing-->
```

```
<cfset teacher = CreateObject('component', 'Teacher').init(test) />
```

```
<cfset teacher.addStudent(anne) />
<cfset teacher.addStudent(bob) />
<cfset teacher.addStudent(carla) />
```

```
<cfoutput>
  #teacher.assignHomework()#<br>
  #teacher.giveTest()#
</cfoutput>
```

```
<!-- now let's have the StudentTeacher -->
```

```
<cfset carla.set('test', test) />
<cfset carla.addStudent(anne) />
<cfset carla.addStudent(bob) />
```

```
<cfoutput>
  #carla.assignHomework()#<br>
  #carla.giveTest()#
</cfoutput>
```

Though it's not important for this discussion, I've included the code for **Test.cfc** for completeness. It can be found in Listing 1.

Object-Based Mixins

Class-based mixins work well when we want to build mixins into our design. As their name suggests, class-based mixins

SiteExecutive[™] Web Content Management

Empower your users to take control of the web

Hundreds of web sites and thousands of people use SiteExecutive to create and manage critical content across commercial, higher education, healthcare and government.

Create and Publish content
Empower non-technical users
Leverage ColdFusion
Automate content migration
Accelerate your business

SYSTEMSALLIANCE
 Think Big. Work Smart



Consolidate your infrastructure and increase performance by leveraging the SunFire[™] x64 Server Family, powered by multi-core or single-core AMD Opteron[™] processors running Solaris[™] OS, Linux or Windows.

Visit www.siteexecutive.com/cfdj or call 877-797-2554

Sun, Sun Microsystems, the Sun logo, Solaris, and Sun Fire are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. SiteExecutive and the SiteExecutive logo are registered trademarks of Systems Alliance, Inc.

affect all instances of the class. Sometimes, though, that's not what we want. We may have objects in memory that we need to make structural changes to; we may need to affect some, but not all, members of a class; or we may be using code that we have no ownership of and no ability to alter.

In any of these cases (and more), a class-based mixin is not appropriate. Object-based mixins provide the ultimate in flexibility — even during runtime. Object-based mixins let us “inject” all methods and variables from one *object* (not *class*) into another object.

Let me offer a very simple example. (The problem with most such simple examples is that without any real-world complexity, we can think of so many other, more straightforward ways than the one presented.) Our example begins with a **Truck.cfc**.

```
<cfcomponent displayName="Truck" extends="BaseComponent">
  <cfset variables.instance.odometer = 0 />

  <cffunction name="init" access="public" output="false">
    <cfargument name="odometer" required="true" />
    <cfset set('odometer', arguments.odometer) />
    <cfreturn this />
  </cffunction>
</cfcomponent>
```

Let's say further that we have several of these objects in memory, but I want to make one of them a guinea pig for a new **GPS** class I've written. The code for **GPS.cfc** is as follows:

```
<cfcomponent displayName="GPS" extends="BaseComponent">
  <cfset variables.instance.locations = null />

  <cffunction name="init" access="public" output="false">
    <cfset var locations = ArrayNew(1) />
    <cfset locations[1] = "Tony's Pizzeria" />
    <cfset locations[2] = "Town library" />
    <cfset locations[3] = "On route to next stop" />
    <cfset locations[4] = "Office" />
    <cfset locations[5] = "Starbucks" />
    <cfset locations[6] = "Casino" />
    <cfset locations[7] = "Gas station" />
    <cfset locations[8] = "Quarry" />
    <cfset variables.instance.locations = locations />
    <cfreturn this />
  </cffunction>

  <cffunction name="locate" access="public" output="false">
    <cfset var locations = get('locations') />
    <cfreturn locations[RandRange(1, 8)] />
  </cffunction>
</cfcomponent>
```

I don't want to change the definition of **Truck** — at least not yet. Instead, I'd prefer to take a single **Truck** object and mix the methods and instance variables of **GPS** into it. Object-based mixins make this very easy, indeed. Here is code to create a **Truck**, a **GPS**, mix the **GPS** into the **Truck** and call a method on that **Truck** that previously belonged only to **GPS**, **locate**:

```
<cfset truck = CreateObject('component', 'Truck').init('Big Red') />
<cfset gps = CreateObject('component', 'GPS').init() />
<cfset truck.mixin(gps) />

<cfoutput>
  #truck.locate()#
</cfoutput>
```

The Universal Superclass

Skeptical readers may be forgiven for examining the **Truck.cfc** only to find that it has no **mixin** method. For that matter, several of the instance variables I used in these examples have been initialized to **null**. But ColdFusion has no **null**. What's going on?

Several OO languages have the notion that classes that do not *explicitly* extend another class *implicitly* extend a single universal superclass. Java has the **Object** class and ColdFusion has **component.cfc**, found in the web root's **WEB-INF/cftags** directory. Set a variable (such as **null**) in this component and it will be available to all components. Write a method (such as **mixin**) in this component and it will be available to all components too.

While adding things directly to **component.cfc** is perfectly fine, I've chosen a slightly different course: I've created a **BaseComponent.cfc**. All components that don't already extend a component in the domain model extend **BaseComponent**, ensuring that all CFCs inherit from **BaseComponent**.

A **BaseComponent** class is the perfect spot to locate our mixin method, which is a model of simplicity:

```
<cffunction name="mixin" access="public" output="false">
  <cfargument name="mixinCode" required="true" />
  <cfset StructAppend(this, arguments.mixinCode) />
  <cfset StructAppend(variables, arguments.mixinCode) />
  <cfset arguments.mixinCode.inject(this) />
</cffunction>
```

In our example of the **Truck** and **GPS**, we call the truck's **mixin** method (which it inherits from **BaseComponent**), passing it the **mixinCode** — the **GPS** object). As part of **mixin**, the **mixinCode**'s **inject** method is called. The **mixin** method adds the methods of the **GPS** to the **Truck**, while the **inject** method adds the private instance variables of the **GPS** to the **Truck**.

“Treating CFCs as if they were a weakened version of Java classes has hidden their real power”

```
<cffunction name="inject" access="public" output="false">
    <cfargument name="mixee" required="true" />
    <cfloop collection="#variables.instance#" item="aKey">
        <cfset arguments.mixee.set(aKey, variables.instance[aKey]) />
    </cfloop>
</cffunction>
```

In Closing

For too long, we've been treating ColdFusion components as if they were a weakened version of Java classes. This (usually unstated) assumption has hidden from us the real power of CFCs. Mixins illustrate how we can find new ways of making software that's more flexible and maintainable. Unlocking the power of CFCs is possible only when we embrace CFCs as they are — and not as poor substitutes for Java.

About the Author

Hal Helms is the author of several books on programming. Hal teaches classes in Java, C#.NET, OO Programming with CFCs, Design Patterns in CFCs, ColdFusion Foundations, Mach-II, and Fusebox. He's the author of the popular Occasional Newsletter and his site is www.halhelms.com.

hal@halhelms.com

Listing 1

```
<cfcomponent displayName="Test" extends="BaseComponent">

    <cfset variables.instance.name = null />

    <cfset variables.instance.percentOfGrade = 0 />

    <cffunction name="init" access="public" output="false">

        <cfargument name="name" required="true" />
        <cfargument name="percentOfGrade" required="true" />

        <cfset set('name', arguments.name) />
        <cfset set('percentOfGrade', arguments.percentOfGrade) />

        <cfreturn this />

    </cffunction>

</cfcomponent>
```

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

Develop Powerful Web Applications with ColdFusion MX 7

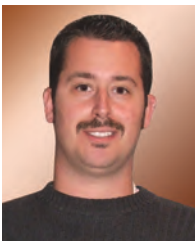
- ▶ Easily build and deploy multi-step data entry forms with CFML tags
- ▶ Solve business reporting problems with Integrated Reporting
- ▶ Manage your account with HostMySite's Control Panel
- ▶ Call 24x7x365 for expert ColdFusion support

Visit hostmysite.com/cfdj for a special offer

 **HostMySite.com**
1-877-215-4678

Introducing objectBreeze

The next generation of ORM



By **Nicholas Tunney**

One complaint I have heard from many developers being introduced to object-

oriented programming (OOP, OO) is that

it is more work than it's worth. Setting up your objects, DAOs, gateways, etc., is a lot

of coding that, until you have realized the power of OO, probably does seem like quite a bit of work. Here is where an ORM tool, such as objectBreeze, steps up to the plate. In the ColdFusion community we have recently seen a few object-relational mapping (ORM) tools hit the scene. These powerful APIs allow a developer to easily interact with their data persistence layer in an object-oriented pattern with a little setup involved (or none in the case of objectBreeze).

Introduction to objectBreeze

The concept behind objectBreeze is very straightforward. By simply installing an instance of the objectBreeze controller on your Web server, any application on the server can instantly access the API with absolutely no setup or configuration. There is no need to create an object class, data access object (DAO),

or gateway. Instead of concentrating on data persistence, you can concentrate on creating your application.

Since objectBreeze instantiates your object from a MasterObject class, there is absolutely no file read/write that occurs when your application accesses the object. objectBreeze is very fast, and since there is no setup involved when using objectBreeze, the learning curve is virtually nonexistent.

To introduce you to objectBreeze, I would like to walk you through creating a sample application. Let's pretend our client, Acme Algorithms, wants to create a search module for their company intranet. Simply put, they want to be able to select a department from a dropdown and/or enter some or all of an employee's name and retrieve a list of employees matching the search criteria. Their database is already designed and we pick out the tables we need for our task (see Figure 1). Seeing these tables makes us instantly think: "We can easily accomplish this with an OO approach." Using OO in this case will allow us to let our search module grow as requirements grow without rewriting large chunks of code.

Step 1: Instantiating the objectBreeze Controller

Since objectBreeze is accessible from any application on your server, you can create several instances of the objectBreeze controller for several database types/datasources in the same application. In this case, we only need one instance of the controller pointing to the client's one datasource. To keep this example simple, we'll instantiate the controller into the application scope in our Application.cfc file (see Listing 1). Then, in the Application.cfc onRequest() method we can set the controller into the variables scope. This will allow us to access objectBreeze from any template's local scope. Outside of this simple example, I would strongly recommend using an MVC pattern and a framework such as

Model-Glue to facilitate this process, but this is outside the scope of this article.

In Listing 1, we first define our database type (current values are MSSQL for Microsoft SQL and MySQL) and our datasource name. We then instantiate the controller and initialize it with the defaults we set above. We now have an instantiated controller object that will allow us to access the powerful features of objectBreeze.

Step 2: Understanding the objectBreeze Controller

There are several helpful methods in the objectBreeze controller. First, objectBreeze provides you with two getters so you can access the database type and datasource name of the current instance of the controller. There are then five methods that allow interaction with objects and controllers, four of which are mentioned below:

```
objectBreeze.createObject("tableName")
```

- `objectCreate()` will return an empty object that contains all the properties of the database table. Remember, objectBreeze requires that the table contain a primary key.

```
objectBreeze.list(table_name, order_by_list)
```

- `list()` returns all records in the table as a `queryObject`. You may send an optional order by statement, which is a list of properties to order by (ASC/DESC permitted).

```
objectBreeze.getByProperty(table_name, property_name, property_value, order_by_list (optional))
```

- `getByProperty()` allows you to specify a single property to query by and returns a query object.

```
objectBreeze.getByWhere(table_name, SQL_where_clause, order_by_list (optional))
```

- `getByWhere()` allows you to send a where statement to the query. This statement should not contain 'WHERE'. You may also specify a list to order by.

`objectCreate()` allows us to create a new object. `List()`, `getByProperty()`, and `getByWhere()` act as an interface to the objectBreeze gateway methods. In this article, we won't be dealing with `collectionCreate()`, but please feel free to read the documentation at <http://www.objectBreeze.com> for more information on using collections in objectBreeze.

Step 3: Understanding the queryObject

The objectBreeze gateway methods return values as a `queryObject`. This object encapsulates the query and allows easy retrieval of the recordset as objects:

```
queryObject.getQuery()
```

- `getQuery()` returns the query as type query.

```
queryObject.getRecordCount()
```

- `getRecordCount()` returns the number of records in the query object.

```
queryObject.hasPrevious()
```

- `hasPrevious()` returns boolean result as to whether or not there is a previous record in the query.

```
queryObject.hasNext()
```

- `hasNext()` returns boolean result as to whether or not there is a next record in the query.

```
queryObject.getPrevious()
```

- `getPrevious()` returns the previous record in the query as an object.

```
queryObject.getNext()
```

- `getNext()` returns the previous record in the query as an object.

As you can see, you may still access the query recordset directly by invoking `getQuery()`. By checking your current position in the query with `hasPrevious()` and `hasNext()`, you can easily loop over the `queryObject`, returning each individual record as an object. This then allows you to compound each object as necessary and output, commit, etc.

Step 4: Interacting with the Object

Before we get down to business with our sample application for Acme Algorithms, we need to understand how an object created with objectBreeze lets us interact with our data persistence layer. In object-oriented programming, we use what is called a Data Access Object (DAO) to retrieve and persist data. This object contains four primary methods: `create()`, `read()`, `update()`, and `delete()`. These methods are mainly referred to as CRUD methods. We can pass an object to any of these four methods to persist, read, or delete records from our data persistence layer. objectBreeze further encapsulates these transactions with the following methods that are built into the object:

```
objectName.getProperty("propertyName")
```

- `getProperty()` returns the value of the specified property

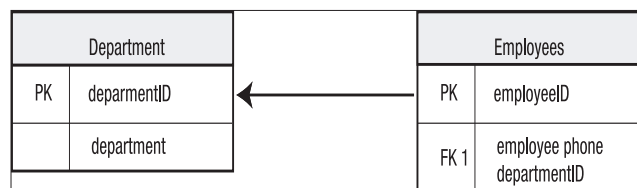


Figure 1

(column) within the object.

```
objectName.setProperty("propertyName", value)
```

- `setProperty()` sets a new value into the specified property (column) within the object.

```
object.getID()
```

- `getID()` returns the value of the primary key property (column) in the object.

```
object.setID("New_PK_Value")
```

- `setID()` sets the value of the primary key property (column) in the object.

```
object.read("priKeyValue")
```

- `read()` loads the supplied object with a record from the database matching a primary key value. If the object contains other objects or collections, they will be read in at this time as well using the parent object's primary key.

```
object.readByProperty()
```

- `readByProperty()` loads the supplied object with a record from the database matching any property values specified in the object. If the object contains other objects or collections, they will be read in at this time as well using the parent object's primary key.

Note: `readByProperty()` can be used in conjunction with the `queryObject`. Once I have loaded my initial object from the `queryObject`, I can specify composition and call `readByProperty()` to load the entire matching composition.

```
object.commit()
```

- `commit()` will either insert or update the object in the database depending on whether or not the primary key value already exists in the database. If you are using an identity field, do not supply a value for the ID. The object will be returned containing the new ID returned from the insert. If the table does not use an identity field, supply the value and it will be created with that value (such as a UUID). If the object contains other objects or collections (composition), these objects will be written to the database as well.

```
object.delete()
```

- `delete()` removes an object from the database by the primary key value defined in the object. This is a hard delete.

Step 5: Understanding Object Composition in objectBreeze

Simply put, object composition is the process of combining several simple objects into one complex object. This can

be very powerful in your application as it allows for faster view creation and simplified object reads and commits. `objectBreeze` facilitates this process with two simple methods located in the object:

```
objectName.containsOne("compounded_table_name")
```

- `containsOne()` creates an object within the existing object.

```
objectName.containsMany("compounded_table_name")
```

- `containsMany()` creates a collection within the existing object.

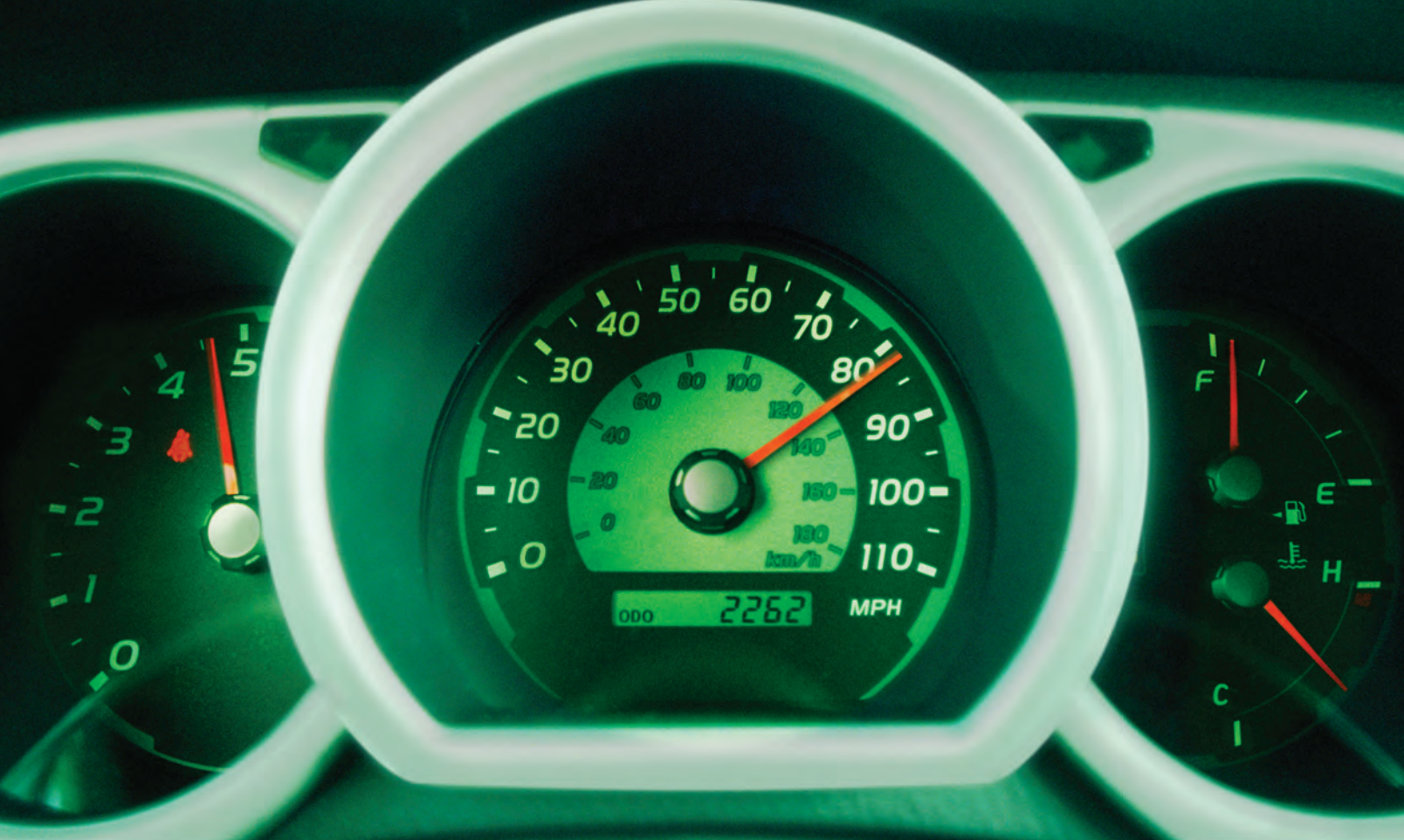
As an example, at Acme Algorithms, departments will have several employees. We can say that departments contain many employees. Instead of coding the old way, which would be to query our database and retrieve all departments joined to the employees table and then grouping the `<cfoutput>` block, returning entirely too much data (duplicate data for department needs to be returned along with each employee record, which is unwieldy and with large applications can chew up memory on your server) or pull all departments and loop over the query, during which you would query the database again for matching employees for the current department and then loop each record in the employee table – using OO, we can create an instance of department (or a collection of department objects), tell `objectBreeze` that the department object contains many employees and then call `department.read()`. This will read the department record and all employees belonging to that department into the composition. A simple loop then quickly allows us to display this object composition. We won't be using this process in our current example, but I encourage you to create your own example using the composition functionality.

Step 6: Putting It All Together

Now that the basic principles are fresh in your mind, let's get back to our sample application for Acme Algorithms. Remember, we are designing a simple search interface to allow intranet users to search employee records by department and employee name. We already decided to use OO so that our module can grow with future requirements, and we decided to use `objectBreeze` due to its simplicity.

Once again, in a real-world application I would recommend using the MVC pattern when developing new functionality, but that is beyond the scope of this article and we will be interacting with `objectBreeze` directly from our view. The first thing we need to design is our search form. Listing 2 is the complete code for our form template (see Figure 2).

What we are doing in Listing 2 is using the `objectBreeze` controller to return a query of all departments in our department table. To accomplish this, we use the `list()` method. Since we don't need to access the individual departments as objects in this template, we can return the query from our `queryObject` using `getQuery()`. Once we have the query, the form is written like any other form.



Rev Up Your Flash Video

Stay Ahead of the Competition With VitalStream and the Enhanced Video Features in Flash 8

With over two years of experience in delivering much of today's most popular media, VitalStream® is the first and most experienced Flash™ video streaming service provider.

Enhanced Flash 8 Video Features:

- New VP6 codec delivers higher quality video at the same bit rate
- 8-bit alpha channel transparency enables you to blend video with other elements
- Improved live video capabilities

VitalStream Complete Toolset for Flash:

- MediaConsole®
- MediaOps™ SDK
- Flash Authentication
- Reporting Dashboard



*Integrate Streaming Media
Into Your Flash Projects*

Take Advantage of the Enhanced Video Features in Macromedia Flash 8
Call (800) 254-7554 or Download Tutorials at www.vitalstream.com/go/mxdj

When we post our form to our display template (see Listing 3), we will be retrieving an object and a queryObject based on our search criteria. There are many different ways to handle this, such as returning a collection of Employee objects, returning two query objects, etc., but for the simplicity of this example, I feel this is the best practice.

The object we create in our template is the department object. We use the read() method in the object class to return a populated object that matches the departmentID, the user specified in the search form. This object contains all of the department properties. If the client would add more properties to the department table, objectBreeze would instantly begin to retrieve those properties as well.

The queryObject we are creating contains all employees that match our search criteria specified in the search form. To get the matching employees, we are using the getByWhere method in the objectBreeze controller. This method lets us specify three arguments:

1. table name
2. where clause (do not include "WHERE")
3. order by list (do not include "ORDER BY", comma-delimited if multiple properties are specified)

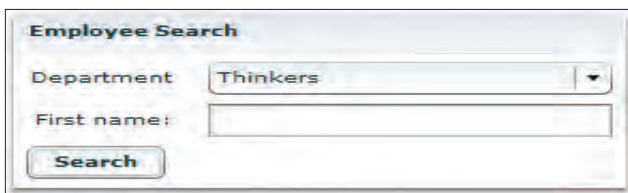


Figure 2

Now that we have our department object instantiated and populated and a queryObject full of matching employees, we can move on to displaying this data to the user. In our column heading we want to output the department name that the user selected. We can do this easily using the class method getByProperty(). getByProperty() will return any property that you specify from the object. In this case, we want to return the department property, so we use getByProperty("department"). We want to ensure that there are employee records to display so we check the recordCount property of the queryObject to make sure records exist, just as we do with a standard CF query. getRecordCount() returns the number of records stored in our queryObject. To understand how we use the queryObject to return objects, review the code in Listing 4.

The <cfloop> tag is looping from 1 to the number of records we have in our queryObject. Since it will stop looping when it


reaches the maximum number of records, we don't need to use the hasNext() method to determine if we have another record to display in this case. Next, we have the queryObject return a populated object using getNext() and set it equal to the employee variable. Just as we did with our department object, we can use the getByProperty() method to return property values from our object. Simple!

Looking Forward

Take time to consider the code. We have just created a very simple search interface that is linked to data persistence (in this example, Microsoft SQL). Note that we did not write a single line of SQL code anywhere in our templates. Since objectBreeze automatically pulls new properties from the tables it persists, if the client decides to add new fields to the employees table, such as birthday or fax, objectBreeze will instantly make that data available to you in your template. All you have to do is pull the additional properties using getByProperty(). If the client decided to allow the user to store multiple e-mail addresses, we would simply need to compound the employee object using containsMany().

This example did not even touch on writing and deleting records. Can you imagine writing the front end for editing the department or employee records? With objectBreeze, you would still not be writing SQL to accomplish this task. A simple commit() will persist the new information for you instantly. You can even commit collections and entire object compositions with one simple line of code.

Summary

I hope this article got you excited about using objects in your ColdFusion applications and showed how easy using objects has become with the new open source ORM tools available to CF developers. I have introduced you to the power of using objectBreeze in a simple application, but objectBreeze does much more than what was discussed in this article. I encourage you to visit www.objectBreeze.com and learn more about how objectBreeze can simplify your ORM. 

About the Author

Nicholas Tunney is a Macromedia Certified ColdFusion developer, and has been programming ColdFusion for over 7 years. He is currently Senior Software Architect for AboutWeb, a consulting firm located in Rockville, Maryland. To learn more about ColdFusion, visit Nic's Blog at <http://www.nictunney.com>.

ntunney@aboutweb.com

Listing 1: Application.cfc

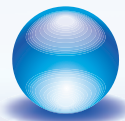
```
<cfcomponent displayName="Application">

    <cffunction name="onApplicationStart" returnType="boolean">
```

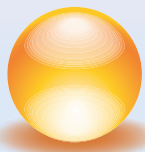
```
        <cfset var appStarted = 1 />
        <cfset application.DBType = "MSSQL" />
        <cfset application.dsn = "communityf" />
        <cfset application.objectBreeze = createObject("component",
            "objectBreeze.assets.cfc.controller.objectBreeze").init(application.
```

H S T I N G . c o m

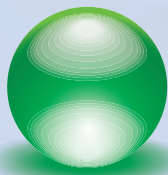
THE FIRST NAME IN HOSTING



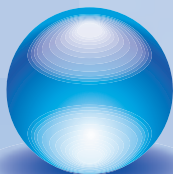
Developers!



Developers!



Developers!



We Love Them!

SERVE. SUPPORT. SECURE. STORE.

www.hosting.com

```
DBType, application.dsn) />
```

```
<cfreturn appStarted>
</cffunction>
```

```
<cffunction name="onRequest" returntype="void">
    <cfargument name="targetPage" type="String" required=true/>

    <cfset variables.objectBreeze = application.objectBreeze />
    <cfinclude template="#arguments.targetPage#">

</cffunction>
</cfcomponent>
```

Listing 2: frm.search.cfm

```
<cfsilent>
    <!--- get a queryObject of all departments in the database --->
    <cfset qoDepartments = objectBreeze.list("Department", "department
ASC") />

    <!--- we do not need to access departments as an object, so let's
get the query instead for simplicity--->
    <cfset qryDepartments = qoDepartments.getQuery() />
</cfsilent>

<cfoutput>
    <cfform action="dsp.results.cfm" format="flash" width="300" time-
out="200">
        <cfformgroup type="panel" label="Employee Search">
            <cfselect name="departmentID" label="Department">
                <!--- loop over our query to populate the select box --->
                <cfloop query="qryDepartments">
                    <option value="#qryDepartments.departmentID#">#qryDep
artments.department#</option>
                </cfloop>
            </cfselect>
            <cfinput id="lastName" name="firstName" maxlength="30"
label="First name:" />
            <cfinput id="btnSubmit" name="btnSubmit" type="submit"
value="Search" />
        </cfformgroup>
    </cfform>
</cfoutput>
```

Listing 3

```
<cfsilent>
    <cfparam name="form.departmentID" default="1" />
    <cfparam name="form.firstName" default="" />

    <!--- create a department object --->
    <cfset department = objectBreeze.createObject("Department") />
    <!--- read into the department object to get the department matching
the id selected on the form --->
    <cfset department.read(form.departmentID) />
    <!--- get a queryObject of employees matching our search criteria
--->
    <cfset qoEmployees = objectBreeze.getByWhere("Employees", "depart-
```

```
mentID = #form.departmentID# AND employee LIKE '#form.firstName#%',
"employee ASC") />
</cfsilent>
```

```
<cfoutput>
    <table>
        <!--- display the department name --->
        <tr>
            <th align="left" colspan="2">Department #department.getProperty(
"department")#</th>
        </tr>
        <!--- if we have records, loop over the matching employees in the
queryObject and display their properties --->
        <cfif qoEmployees.getRecordCount()>
            <cfloop from="1" to="#qoEmployees.getRecordCount()#" index="i">
                <!--- retrieve the next record as an object --->
                <cfset employee = qoEmployees.getNext() />
                <!--- display the employee --->
                <tr>
                    <td align="left">#employee.getProperty("employee")#</td>
                    <td align="left"><cfif len(employee.getProperty("phone"))>
#employee.getProperty("phone")#<cfelse>N/A</cfif></td>
                </tr>
            </cfloop>
        <!--- if we had no records, display a friendly message --->
        <cfelse>
            <tr>
                <td align="left" colspan="2">
                    No records matched your search criteria.
                </td>
            </tr>
        </cfif>
    </table>
</cfoutput>
```

Listing 4

```
<cfloop from="1" to="#qoEmployees.getRecordCount()#" index="i">
    <!--- retrieve the next record as an object --->
    <cfset employee = qoEmployees.getNext() />
    <!--- display the employee --->
    <tr>
        <td align="left">#employee.getProperty("employee")#</td>
        <td align="left"><cfif len(employee.getProperty("phone"))>#employ
ee.getProperty("phone")#<cfelse>N/A</cfif></td>
    </tr>
</cfloop>
```

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

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



Java for ColdFusion Programmers?

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

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

CFDJ Advertiser Index

ADVERTISER	URL	PHONE	PAGE
AJAX SEMINAR	WWW.AJAXSEMINAR.COM	201-802-3022	49
CFDYNAMICS	WWW.CFDYNAMICS.COM	866-233-9626	4
CFDJ	HTTP://COLDFUSION.SYS-CON.COM/	800-303-5282	47
COMMUNITYMX	WWW.COMMUNITYMX.COM/TRIAL		51
ECLIPSE	HTTP://ECLIPSE.SYS-CON.COM	888-303-5282	39
EDGEWEBHOSTING	EDGEWEBHOSTING.NET	1-866-334-3932	9
HALHELMS	HALHELMS.COM		31
HOSTING.COM	WWW.HOSTING.COM		29
HOSTMYSITE	WWW.HOSTMYSITE.COM	877-215-4678	19, 23
INTERAKT	WWW.INTERAKTONLINE.COM/		3
INTERMEDIA	WWW.INTERMEDIA.NET	888-379-7729	52
ITSG		201-802-3021	48
JDJ	WWW.JDJ.SYS-CON.COM	1-888-303-5282	43
MACROMEDIA	MACROMEDIA.COM/GO/CFMX7_DEMO	415-252-2000	2
MACROMEDIA	WWW.MACROMEDIA.COM/GO/8_STUDIO8	415-252-2000	11
PAPERTHIN	WWW.PAPERTHIN.COM	800-940-3087	13
SYSTEMS ALLIANCE	WWW.SITEEXECUTIVE.COM/CFDJ	877-797-2554	21
VITALSTREAM	WWW.VITALSTREAM.COM	800-254-7554	27
WEBAPPCABARET	WWW.WEBAPPCABARET.COM/JDJ.JSP	866-256-7973	6
WEBAPPER	WWW.SEEFUSION.COM		41
WSEGE/OPEN SOURCE	EVENTS.SYS-CON.COM	201-802-3066	37

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 agency 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.

REPRINT

Once you're in it...



...reprint it!

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

Contact Dorothy Gil
201 802-3024
dorothy@sys-con.com

REprints

SYS-CON MEDIA



2005 CFDJ

READERS' CHOICE AWARDS

Winners and runners-up in 12 award categories



BEST BOOK

Winner:

ColdFusion MX Web Application Construction Kit

BY BEN FORTA, NATE WEISS, LEON CHALNICK, ANGELA C. BURAGLIA (MACROMEDIA PRESS)

Runners-Up:

- 1) Programming ColdFusion MX
(BY ROB BROOKS-BILSON (O'REILLY))
- 2) Advanced Macromedia ColdFusion MX Application Development
(BY BEN FORTA AND SARGE SARGENT (MACROMEDIA PRESS))
- 3) Dreamweaver MX 2004: The Missing Manual
(BY DAVID MCFARLAND (POGUE PRESS))

BEST WEB SITE OR COMMUNITY

Winner:

Macromedia DevCenter
(MACROMEDIA)

Runners-Up:

1. EasyCFM.com
(EASYCFM)
2. CommunityMX.com
(COMMUNITY MX)
3. TeraTech CFConf Central
(TERATECH, INC.)

BEST CONTENT MANAGEMENT TOOL

Winner:

KTML – Online Visual HTML Editor
(INTERAKT)

Runners-Up:

1. Macromedia Contribute
(MACROMEDIA)
2. FarCry CMS
(DAEMON)
3. CommonSpot Content Server
(PAPERTHIN, INC.)

MOST INNOVATIVE APPLICATION

Winner:

MX Kollection
(INTERAKT)

Runners-Up:

1. Macromedia Studio MX 2004
(MACROMEDIA)
2. BlueDragon for the Microsoft .NET Framework
(NEW ATLANTA COMMUNICATIONS)
3. FarCry CMS
(DAEMON)

BEST CF WEB SERVICES

Winner:

CF Tips
(WWW.FORTA.COM/BLOG)

Runners-Up:

1. House of Fusion RSS Feeds
(HOUSE OF FUSION)
2. Fullasagoog.com
(GEOFF BOWERS)
3. ActivSpell
(ACTIVSOFTWARE INC.)

BEST CONSULTING SERVICE

Winner:

TeraTech Consulting
(TERATECH, INC.)

Runners-Up:

1. Fig Leaf Software, Inc.
(FIG LEAF SOFTWARE, INC.)
2. Codesweeper
(CODESWEOPER)
3. ColdFusion Web Hosting and Custom Application Development
(ANGRY SAM PRODUCTIONS)

BEST CUSTOM TAG OR COMPONENT

Winner:

MX Widgets
(INTERAKT)

Runners-Up:

1. KTML – Online Visual HTML Editor
(INTERAKT)
2. TeraTech CFXGraphicsserver
(TERATECH, INC.)
3. jComponents
(CLEARSOFTWARE.NET)

BEST EBUSINESS SOFTWARE

Winner:

MX Shop (InterAKT)

Runners-Up:

1. Macromedia Flex
(MACROMEDIA)
2. FuseTalk
(FUSETALK INC.)
3. Verabase Pro
(STANDPIPE STUDIOS, L.L.C.)



BEST EDUCATION AND TRAINING

Winner:

Macromedia Captivate
(FORMERLY ROBODEMO) (MACROMEDIA)

Runners-Up:

1. TeraTech Training
(TERATECH, INC.)
2. CFMX Exam Buster
(CENTRASOFT CORP.)
3. Fig Leaf Software Certified Macromedia Training
(FIG LEAF SOFTWARE, INC.)

BEST WEB APPLICATION

Winner:

MX Shop
(INTERAKT)

Runners-Up:

- 1) FarCry CMS
(DAEMON)
- 2) FuseTalk
(FUSETALK INC.)
3. Verabase Pro
(Standpipe Studios, L. L. C.)

BEST WEB DEVELOPMENT TOOL

Winner:

Macromedia Studio
(MACROMEDIA)

Runners-Up:

1. Dreamweaver
(MACROMEDIA)
2. MX Kollection
(INTERAKT)
3. CFEclipse
(TIGRIS)

BEST WEB HOSTING

Winner:

Edgewebhosting.net Managed/ Dedicated/Colocation ColdFusion Hosting
(EDGEWEBHOSTING.NET)

Runners-Up:

1. HostMySite.com's ColdFusion Web Hosting
(HOSTMYSITE.COM)
2. ColdFusion Hosting
(CFDYNAMICS)
3. ColdFusion Web Hosting
(CRYSTALTECH)

Completing The Real Estate Sample Application

Managing property listings through Flash Forms

PART 2



By Laura Arguello
and Nahuel Foronda

In Part 1 of this tutorial (CFDJ, Vol.7, issue 12), you built a search interface for the Real Estate sample application. In Part 2, you'll learn how to populate a form by binding the fields to a datagrid, and then edit, add, and remove records from the database. You will add functionality to the one-screen interface you started to build in Part 1 using ColdFusion and Flash Remoting.

Requirements

To complete this tutorial you will need to install the following software and files:

ColdFusion MX 7.01

For a trial download go to http://www.macromedia.com/cfusion/tdrc/index.cfm?product=coldfusion&promoid=devcenter_tutorial_product_090903. To buy go to http://www.macromedia.com/software/coldfusion/buy/?promoid=devcenter_tutorial_coldfusion_090903. To get the updater go to <http://www>.

This article originally appeared on the Adobe Developer Center. Reprinted with permission.

macromedia.com/support/coldfusion/downloads_updates.html#mx7.

- **Database:** Microsoft Access, SQL Server, or MySQL
- **Tutorials and sample files:** http://download.macromedia.com/pub/developer/realestate_pt2.zip. This tutorial builds on the files from Part 1 (http://coldfusion.sys-con.com/read/172578_f.htm or <http://www.macromedia.com/devnet/coldfusion/articles/flashforms.html>). You can use your files from there and add the functionality from this tutorial. If you prefer, you can download this file, which has the complete code for Part 2.

To install the sample application for Part 2, unzip the files, copy the realestate folder to your Web root, create a data source called **realEstate** in the ColdFusion Administrator, and then browse to <http://localhost/realestate/index.cfm> or <http://localhost:8500/realestate> for ColdFusion standalone installations. See the full instructions in the Readme file.

Prerequisite Knowledge

- Part 1 of this tutorial
- Basic knowledge of ColdFusion components and Flash Forms
- Ability to set up a data source and write simple SQL statements

Completing the Real Estate Management System Sample Application

In this tutorial, you'll populate the Add/Edit panel with the record details for the user's search so that the user can edit, remove, and add new records from the same panel (see Figure 1).

Code the following functionality:

1. Populate the edit form by binding fields to the selected

record in the datagrid.

2. Create a ColdFusion component to add, update, and delete records from the database.
3. Create a Flash Remoting service CFC.
4. Call the service and get results.

Populating the Edit Form

The bottom-right panel contains a form with an input for every attribute of a property listing: address, price, number of bedrooms, bathrooms, amenities, and so forth. Most of the fields are text input fields, but there are also controls for selects, radio buttons, check boxes, a text area, and a date field.

When a user opens the application (you can try it by going to <http://localhost:8500/realstate/index.cfm>), he or she selects search criteria and the application returns the search results. The user clicks one of the records in the datagrid and expects the detail panel in the bottom-right corner to display the details of the selected record. Where will the necessary information come from?

In Part 1 of this tutorial, the search returned query results that contained all of the columns in the “listing” table in the database. However, only a few of those columns – namely Price, Type, Bedrooms, Bathrooms, and Footage – appeared in the datagrid.

“What a waste of bandwidth!” you may have thought. Not really so, because you will need the additional columns to populate the entire form.

Populate each field from a column of the query returned by the search. The query that populates the datagrid is “stored” in the datagrid. To get the data corresponding to the selected row in the grid, access the cffgrid by its name (listingGrid) and its special property (selectedItem); the name and property specify the row the user selected.

Then, with dot notation, select the specific column you need, which is different for each input. The complete path is this: listingGrid.selectedItem.columnName. Use this path for all of the columns of the query, even if they were not defined as datagrid columns with the cffgridcolumn tag.

Binding Each Field Type

Now that you know how to get the data for each property listing attribute, you are ready to display it. Just below the search results, add a new panel (note that Part 1’s sample files already contain this panel) that contains the add/edit form:

```
<cfformgroup type="panel" label="Add / Edit Properties">
</cfformgroup>
```

Within this panel, you will add, one by one, all the fields corresponding to each property listing attribute, such as address, number of bedrooms, and so forth. Each field has a different length or type, so you will use different controls or variations of the same controls as user inputs. There are some differences in the way you populate each control.

Populating *text input* controls (<cfinput type="text">), *textarea* controls (<cftextarea>) and *datefield* controls (<cfinput type="datefield">) is straightforward if you simply use their bind property:

```
bind="{listingGrid.selectedItem.columnName}"
```

Checkbox values can be true or false. The column that populates

the *checkbox* control must be a Boolean type so that you can write:

```
value="{listingGrid.selectedItem.columnName}"
```

Select and *radio button* controls are a different challenge. Neither of them have a bind or similar attribute that you can use to bind them directly. Instead, you must make the controls acquire the right choice when the selected record in the datagrid changes.

In ActionScript, controls fire events when certain things happen. For example, the datagrid fires an onChange event each time the user selects an item. You can make your cffgrid call a function or run a piece of ActionScript code every time that event fires. To tell the datagrid what to do in that event, use the onChange attribute:

```
<cffgrid name="listingGrid" onchange="listingGridChanged()">
```

You can call any function you want, either a built-in function such as alert(‘Datagrid selection changed!’) or a custom function.

If you choose to call a custom function, such as listingGridChanged(), you must declare it in a <cfformitem type="script"> block:

```
<cfformitem type="script">
public function listingGridChanged():Void {
    //select item in dropdowns
    //select item in radio button group
}
</cfformitem>
```

To avoid repeating the same code for every select control, create a helper function to select a specific item in the select control. This function looks for the option value that matches the desired item and then selects it:

```
public function selectOption(select: mx.controls.ComboBox, optionToSelect:
String):Void {

    //go through every record to find the right one
    for (var i:Number = 0; i< select.length; i++) {
        if (select.getItemAt([i]).data == optionToSelect){
            select.selectedIndex = i;
            break;
        }
    }
}
```

Figure 1: Complete Real Estate Management application

```

    }
}

```

Last, to select a specific value in a *radio button* group, use the `selectedData` property:

```
myRadioButton.selectedData = listingGrid.selectedItem.columnName;
```

Putting everything together, the complete `listingGridChanged()` function looks like the following:

```
function listingGridChanged():Void {
    // select the matching state
    selectOption(edit_state, listingGrid.selectedItem.state);

    // select the matching property type
    selectOption(edit_type, listingGrid.selectedItem.type);

    //select the matching choice in the radio button
    //for status
    edit_status.selectedData = listingGrid.selectedItem.status;

    //after selecting a radio button, it acquires focus, so bring focus
    back to grid
    listingGrid.setFocus();
}
```

Now the edit panel is correctly populated with the data in the selected row and the user can edit it and submit it to the server.

Note that if you use `listingGrid.selectedItem.columnName` directly, and there is no item selected in the datagrid, your controls may show undefined as their value. To avoid that, use the if-else shorthand statement to check whether the user has selected an item in the datagrid. If the user has selected an item, show the value; otherwise, show an empty control. For example, the city text input looks like the following:

```
<cfinput type="text" name="edit_city" bind="{(listingGrid.selectedItem!=undefined)?listingGrid.selectedItem.city:''}" label="city" />
```

Adding, Updating, and Removing Records

Before continuing with the user interface, you must prepare the components that handle the insert, update, and delete queries. Just as you did for the search functionality, you must write two components – one that interacts with the database and one that acts as a service waiting for Flash Remoting calls.

First, create the first component by making a file called **ListingDAO.cfc** and place it in the `\realestate\components\` directory.

Because the ListingDAO component will handle all the queries (insert, update, and delete), it contains four functions: create, update, delete, and fetch – a function to get a specific record from the database. Create and update functions will receive a set of arguments representing each listing attribute, such as address, number of bedrooms, or square footage. Delete and fetch require only the record's primary key.

In ListingDAO.cfc, add the code in Listing 1, inserting all the arguments, completing the query, and implementing the other four functions:

The Fetch function takes an `mls_id` and returns a structure with the corresponding property listing matching the `mls_id`. This structure will later be used to populate the created or updated records in the datagrid. Refer to the sample files you downloaded in the Requirements section to see the complete code for all the functions in this component.

Creating the Flash Remoting Service

Flash Remoting connects the user interface with the component you just wrote. Now you need to expose the component's function as Flash Remoting services so that the user can actually change data in the database. You do not need to write the service component from scratch; you can simply add the necessary functions to the ListingService.cfc component file described in Part 1 of this tutorial.

You must add functions to handle an insert request (`<cffunction name="create">`), an update request (`<cffunction name="update">`), and a delete request (`<cffunction name="remove">`). As with the search function, you can directly pass the same arguments to the ListingDAO component. When using functions as Flash Remoting services, set their access attribute as "remote" (`access="remote"`).

Add the code in Listing 2 to the ListingService.cfc from Part 1 or use the ListingService.cfc file included in the sample files in the Requirements section in this tutorial.

These newly added functions return a structure with a "status" field, indicating whether the operation was successfully performed, and an "item" field to return the updated or created item. You may wonder why it is necessary to send the item back to the client interface. The reason is that any Flash Remoting call is asynchronous, which means that when your application returns the result of a record insert or update, the user may have changed some inputs or the datagrid selected item – you cannot rely on the data in the form fields at that time.

Calling the Service and Handling the Result

At the bottom of the edit form in the index.cfm file, there is a button to submit the form:

```
<cfinput
type="button"
name="editBtn"
onclick="submitEdit()"
value="{(listingGrid.selectedItem == undefined)?'Submit':'Apply Changes'}"
/>
```

You may have noticed that it's not an input-type submit. Just as with the search panel, the submit button is implemented with a simple button that calls a function when it's clicked (`onclick="submitEdit()"`). In addition, its label is dynamic and changes depending on whether a user has selected an item in the datagrid, which indicates the user is editing a record; or nothing is selected, indicating that the user is entering a new record.

To change this label, use the value attribute and a binding with

ENGAGE AND EXPLORE...

The Technologies, Solutions and Applications that
are Driving Today's Initiatives and Strategies...

CALL FOR PAPERS NOW OPEN!

SOA 10th International WebServices Edge conference+expo 06



June 2006 | New York, NY

The Sixth Annual SOA Web Services Edge 2006 East - International Web Services Conference & Expo, to be held June 2006, announces that its Call for Papers is now open. Topics include all aspects of Web services and Service-Oriented Architecture

Suggested topics...

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

CALL FOR PAPERS NOW OPEN!

2006 ENTERPRISE> OPENSOURCE CONFERENCE+EXPO



June 2006 | New York, NY

The first annual Enterprise Open Source Conference & Expo announces that its Call for Papers is now open. Topics include all aspects of Open Source technology. The Enterprise Open Source Conference & Expo is a software development and management conference addressing the emerging technologies, tools and strategies surrounding the development of open source software. We invite you to submit a proposal to present in the following topics. Case studies, tools, best practices, development, security, deployment, performance, challenges, application management, strategies and integration.

Suggested topics...

- > Open Source Licenses
- > Open Source & E-Mail
- > Databases
- > ROI Case Studies
- > Open Source ERP & CRM
- > Open-Source SIP
- > Testing
- > LAMP Technologies
- > Open Source on the Desktop
- > Open Source & Sarbanes-Oxley
- > IP Management

Submit Your Topic Today! events.sys-con.com

Sponsored by

WebServices
JOURNAL

XML JOURNAL

NET JOURNAL

eclipse) developer's journal

WebSphere
JOURNAL

Information
STORAGE+SECURITY
JOURNAL

wildj

JDJ

LinuxWorld

MX
developer's journal

asp.net PRO

SDTimes

CoDe

Software Test
& Performance



Attention Exhibitors:

An Exhibit-Forum will display leading Web services and OpenSource products, services, and solutions

*Call for Papers email: events@sys-con.com

For Exhibit and Sponsorship Information - Call 201 802-3023

Produced by **SYS-CON**
EVENTS

© 2005 WEB SERVICES EDGE. ALL RIGHTS RESERVED

a shorthand if-else statement that tests for `listingGrid.selectedItem == undefined`, meaning no item was selected. In that case, the value becomes “Submit” for new records; otherwise the value becomes “Apply Changes” for a record that’s being updated.

You’ll also declare and implement the `submitEdit()` function in a `<cfformitem type=”script”>` block in `index.cfm`. Following the same method used to submit the search form, gather the data contained in each input, radio button, select, text area, and check box to send to the service method. Unlike the search panel, you must validate this form before submitting it.

There are several required fields and fields that accept only numbers, zip codes, and other validation types. When a user submits a form using a `<cfinput type=”submit”>` tag, Flash Forms run the validation automatically and don’t submit the form until it validates entirely. Because you don’t use a submit button, but a regular button, the form does not automatically run this validation. To ensure that you validate the form before sending the data to the Flash Remoting service, use the following code to test it:

```
if( mx.validators.Validator.isStructureValid(this, 'RealEstateAdmin') ){
    //make the call
}
```

Add this test right before sending the data to the Flash Remoting service in the `submitEdit()` function of the `index.cfm` file. If `mx.validators.Validator.isStructureValid(this, 'RealEstateAdmin')` returns true, it means the form is valid and you can proceed with the call.

Recall that `RealEstateAdmin` is the name of the form. Change it according to your own form name. Calling this function indicates whether there are validation issues; it also activates the red error tips and red border on the fields so that the application informs the user about what he or she needs to complete or modify.

Last, because you use the same button to submit a new record and an updated record, the function must know what service method to call (update or create). For simplicity’s sake, check for the grid selected item – the same test you did to change the submit button’s label value. A more complex approach would be to set a global variable that indicates whether the user is in “edit” or “add” mode. The sample files use a global variable called `isEditMode`, which is set when the user changes the selection in the datagrid.

You make the Flash Remoting service call in the `submitEdit()` function using the same service connection object you created in Part 1 that was stored in a global object and called:

```
RealEstateAdmin.myGlobalObjects.listingService
```

The function you create to handle the submit gathers the form data from each field as it was described in Part 1 when gathering the Search panel data (see Listing 3).

After making the call, the server returns a successful response. But your form is not yet ready to handle that response.

If you recall from Part 1, you wrote a function within your form called `setUpRemoting()` that created the Flash Remoting service connection. When you created this connection, you had to supply an object to be called when the server returns a response, as follows:

```
RealEstateAdmin.myGlobalObjects.listingService = connection.getService("D
evnetRealState.services.ListingService", responseHandler );
```

The object called when the server returns a response is the `responseHandler` object. When you created it, you also implemented two functions: the `onResult` and `onStatus` functions. If you don’t modify this object and use this instance of the handler object to call the update function, the `onResult` function of `responseHandler` object will be called when the server sends the response. This is because the `onResult` function is the default function that Flash Remoting uses if you specify nothing else.

The problem is that you used the `onResult` function to set the datagrid’s `dataProvider` with the returned results. That is not preferable for all the other service calls. To handle those calls, you must implement other functions in the `responseHandler` object. When you don’t wish to use the two default functions (the `onResult` and `onStatus` functions), you must create one function for each service method you would like to call.

The function name should be the name of the service method followed by `_Result`. For example, if you want to call the update method, you will need to add a function called `update_Result()` to the handler object (`responseHandler` object in this example):

```
// handle create response
responseHandler.create_Result = function( results: Object ):Void {
}

// handle update response
responseHandler.update_Result = function( results: Object ):Void {
}

// handle remove response
responseHandler.remove_Result = function( results: Object ):Void {
}
```

Next you’ll see how to implement these functions.

Adding a New Record

The `create()` method of the service CFC (`realestate/services/ListingService.cfc`) inserts a new property listing into the database by calling the `create()` method in the `ListingDAO.cfc` (`realestate/components/ListingDAO.cfc`). Then the CFC returns a structure with status, message, and item keys.

If the CFC was able to insert the item successfully, it returns the status field as true and an item field containing the complete newly inserted record. This is useful not only if the data in the form fields changes while the service responds but also to get any field that may have changed or may have been calculated on the server.

What do you think the `create_Result()` function does with the data the service returns? You probably guessed that you need to add this new record to the datagrid so that it’s available for editing. For simplicity’s sake, add it to the top of the grid (only if the returned status is true) using the `addItemAt()` function of the datagrid. This function takes the index where to insert the item and the actual item to insert. Because the CFC returned the complete item, you can supply it directly to the datagrid by getting

eclipse) developer's journal

A DIGITAL PUBLICATION

Your One-Stop Guide to the Eclipse Ecosystem

The main purpose of *Eclipse Developer's Journal (EDJ)* is to educate and inform users of Eclipse and developers building plug-ins or Rich Client Platform (RCP) applications. With the help of great columnists, news you can use, and technical articles, *EDJ* is a great information source that you will be able to use to educate yourself on just about anything Eclipse related

Subscribe Today!

**FOR YOUR DIGITAL
PUBLICATION**

(AVAILABLE IN DIGITAL FORMAT ONLY)

6 Issues for \$19.99

Visit our site at www.eclipse.sys-con.com or
call 1-888-303-5282 and subscribe today!

OFFER SUBJECT TO CHANGE WITHOUT NOTICE



<http://eclipse.sys-con.com>

SYS-CON.COM

SYS-CON Media, the world's leading publisher of *i*-technology magazines for developers, software architects, and e-commerce professionals, brings you the most comprehensive coverage of Eclipse

it from the results object and passing it as the second addItemAt() parameter:

```
listingGrid.addItemAt(0,results.item);
```

This following is the complete create_Result() function:

```
responseHandler.create_Result = function( results: Object ):Void {
    //only add the item to the datagrid if creation was successful
    if (results.status){
        //add an item at the top of the grid, containing
        //the item that was returned in the "item" field
        //of the results
        listingGrid.addItemAt(0,results.item);
        alert("Item inserted");
    }
    else {
        alert("Item could not be inserted");
    }

    //remove the clock cursor
    mx.managers.CursorManager.removeBusyCursor();
}
```

Updating a Record

The update_Result() function that you need to implement in the responseHandler object to handle the results of a call to the update() service method is similar to the create_Result() method. In this case, however, edit the field that has been updated instead of adding a new item to the datagrid.

The problem with asynchronous calls is that you don't know when the results will return and whether the form will be in the same state as when the user made the original call. You cannot assume that the updated record is still the grid's selected item. Just as you created a helper function to select items in a select control (with the cfselect tag), you can create a helper function that finds an item by the ID (mls_id in this case) in the datagrid, and returns its index. Then you will be able to edit the correct item.

Write this function inside a <cfformitem type="script"> block as follows:

```
public function findItem(grid:mx.controls.DataGrid, columnToSearch:String,
id:String):Number {

    for (var i = 0; i < grid.dataProvider.length; i++){
        if (grid.getItemAt(i)[columnToSearch] == id){
            //found
            return i;
        }
    }
    //not found
    return -1;
}
```

When the service returns the results, you look for the item updated in the grid and change it using replaceItemAt() function of the datagrid. This function takes the index of the record to edit and the new value to assign to the row. Again, you can supply the returned item as the new value to the replaceItemAt() function (see Listing 4).

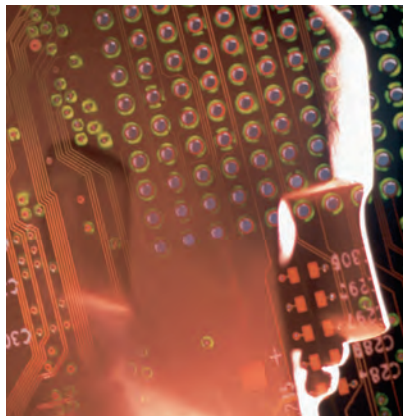
Deleting a Record

When a user selects an item in the datagrid, he or she can also remove it from the database by clicking on the Remove Property button. Using the same technique as you used for the other buttons, call a function when the user clicks the Remove Property. You will name this function submitRemove():

```
<cfinput type="button" name="removeBtn" value="Remove Property"
onclick="submitRemove()" />
```

The service connection calls the remove function on the service CFC to delete items. That function only requires the mls_id of the item to remove, so you need only get that field from the form and send it. The field is an input called edit_mls_id, so you'll access its current data through edit_mls_id.text:

```
public function submitRemove():Void {
    <!-- call service, sending the id -->
    RealEstateAdmin.myGlobalObjects.listingService.remove(edit_mls_id.text);
}
```



As with the other functions, put it into a <cfformitem type="script"> block.

When the service sends a successful response, the responseHandler object that receives the response must remove the item from the data grid, because it no longer exists in the database. Because you don't know which row holds this item, you can use the same helper function, findItem(), to look for the item to delete.

Next, use the removeItemAt() function to remove the item from the datagrid. Of course, you would only do that if the deletion was successfully performed, given by the status field of the results object returned by the CFC:

```
responseHandler.remove_Result = function( results: Object ):Void {
    if (results.status){
        listingGrid.removeItemAt( _root.findItem(listingGrid, 'mls_id',
results.item.mls_id));

        //show the message
        alert("Item removed");
    }
    else {
        //show the message
        alert("Item could not be removed");
    }
}
```

```
mx.managers.CursorManager.removeBusyCursor();
}
```

That's it! You have now the complete application with add, update, and delete functionality ready to use.

Where to Go from Here

You can enhance interfaces greatly using Flash Forms and Flash Remoting. Although limited in some respects, Flash Forms can help you create more responsive and usable forms. Using this sample application as an example, you can apply the ideas shown here to your own applications.

As you have seen in this tutorial, you can use ActionScript to leverage the power of Flash Forms further. If you're feeling adventurous, you can also explore Flex (www.macromedia.com/software/flex) as a platform for creating full-featured RIAs.

Listing 1

```
<cfcomponent name="ListingDAO">

    <cffunction name="create" output="false" hint="Inserts a new record
    and returns the primary key of the inserted item" access="public"
    returnType="string">
        <cfargument name="mls_id" required="true" type="string"
        hint="Primary key"/>
        <cfargument name="address" required="true" type="string"
        hint="Address"/>
        <cfargument name="city" required="true" type="string" hint="City"/>
        <!-- additional arguments removed -->

        <cfset var insertListingQuery = "" />

        <cfquery name="insertListingQuery" datasource="realestate">
            INSERT INTO listing
            (mls_id, address, city
            <!-- additional columns removed -->
            )
            VALUES (
                <cfqueryparam value="#arguments.mls_id#" cfsqltype="CF_SQL_VAR-
                CHAR" maxlength="10"/>,
                <cfqueryparam value="#arguments.address#" cfsqltype="CF_SQL_
                VARCHAR" maxlength="100"/>,
                <cfqueryparam value="#arguments.city#" cfsqltype="CF_SQL_VAR-
                CHAR" maxlength="50"/>
                <!-- additional values removed -->
            )
        </cfquery>

        <cfreturn arguments.mls_id>

    </cffunction>

    <!-- additional functions removed:
    <cffunction name="update" returnType="string">
```

About the Authors

Laura Arguello is one of the founders of Blue Instant (www.blueinstant.com), where she has been creating Web applications for the last five years. Apart from her company, she also maintains a blog, AS Fusion (www.asfusion.com), where she and Nahuel Foronda write about ColdFusion, Flash, and other Macromedia and Web technologies.

Nahuel Foronda is the founder and lead Flash developer for Blue Instant (www.blueinstant.com), a Web development firm specializing in Rich Internet Applications. During his five years of experience with Flash, he has created award-winning applications and Web sites. He also maintains a blog on Flash and ColdFusion called AS Fusion (www.asfusion.com).

```
<cffunction name="remove" returnType="string">
<cffunction name="fetch" returnType="struct">
    ---
</cfcomponent>
```

Listing 2

```
<cffunction name="create" access="remote" returnType="struct"
output="false" description="Creates a property listing">
<cfargument name="mls_id" required="true" type="string" hint="Primary
```



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

Now you can.

Monitor and troubleshoot your ColdFusion applications and servers in real-time with SeeFusion.



Key Features:

- * View metrics for request times, queries, throughput, memory utilization, server uptime, and other critical data
- * Attach debugging output to pages
- * Log metrics to a database for in-depth analysis
- * Stop long-running/hung requests
- * Easily monitor multiple servers at once
- * Connect to notification systems via XML bridge
- * And more!

www.seefusion.com

SeeFusion is a product of Webapper Services, LLC - Development, Consulting, Products
www.webapper.com

```

key"/>
<cfargument name="address" required="true" type="string"
hint="Address"/>
<cfargument name="city" required="true" type="string" hint="City"/>
<!-- additional arguments removed -->

<!-- you could also use cfinvoke or access a component in memory
--->
<cfset var listingManager = createObject("component", "realestate.
components.ListingDAO")/>

<cfset var result = structnew()/>

<!-- insert record --->
<cfset listingManager.create(argumentCollection=arguments) />

<!-- retrieve the complete record to show.
fetch() returns a structure with the query columns as keys
--->
<cfset result["item"] = listingManager.fetch(arguments.mls_id) />

<!-- set successful operation status
(You should set this status as true or false depending
on whether the operation was really successful or not)
--->
<cfset result["status"] = javacast("boolean", true) />
<cfreturn result/>

</cffunction>

```

Listing 3

```

public function submitEdit():Void {
    //declare an object that will be sent as a structure to the
    service
    var editArguments = {};

    //to get data contained in Text inputs,
    //textareas and dateFields use myTextInputName.text
    editArguments.address = edit_address.text;
    editArguments.listedOn = edit_listedOn.text;
    editArguments.remarks = edit_remarks.text;

    //to get data from radio buttons,
    //use myRadioButtonName.selectedData
    editArguments.status = edit_status.selectedData;

    //checkboxes store their true/false value in
    //their selected property myCheckboxName.selected
    editArguments.hasPool = edit_hasPool.selected;

    //to get the selected choice in a dropdown,

```

```

    //use mySelectName.value or mySelectName.selectedItem.data
    editArguments.state = edit_state.value;

    //run validation
    if( mx.validators.Validator.isStructureValid(this, 'RealEstateAd-
min') ){
        //show clock cursor
        mx.managers.CursorManager.setBusyCursor();

        //this is a new record
        if (listingGrid.selectedItem == undefined) {
            //call create service method
            RealEstateAdmin.myGlobalObjects.listingService.create(
editArguments);
        }

        //this is an updated record
        else {
            //call update service method
            RealEstateAdmin.myGlobalObjects.listingService.update(
editArguments);
        }
    }
}

```

Listing 4

```

responseHandler.update_Result = function( results: Object ):Void {
    var item:Object = results.item;
    //find the item's index
    var index:Number = _root.findItem(listingGrid,'mls_id',item.mls_id);

    if (results.status && index >= 0){
        listingGrid.replaceItemAt(index,item);

        //show a message
        alert("Item updated");
    }
    else {
        //show a message
        alert("Item could not be updated");
    }

    mx.managers.CursorManager.removeBusyCursor();
}

```

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

The World's Leading Java Resource Is Just a >Click< Away!

JDJ is the world's premier independent, vendor-neutral print resource for the ever-expanding international community of Internet technology professionals who use Java.



ONLY
\$69⁹⁹
ONE YEAR
12 ISSUES

**Subscription Price Includes
FREE JDJ Digital Edition!**

www.JDJ.SYS-CON.com

or **1-888-303-5282**



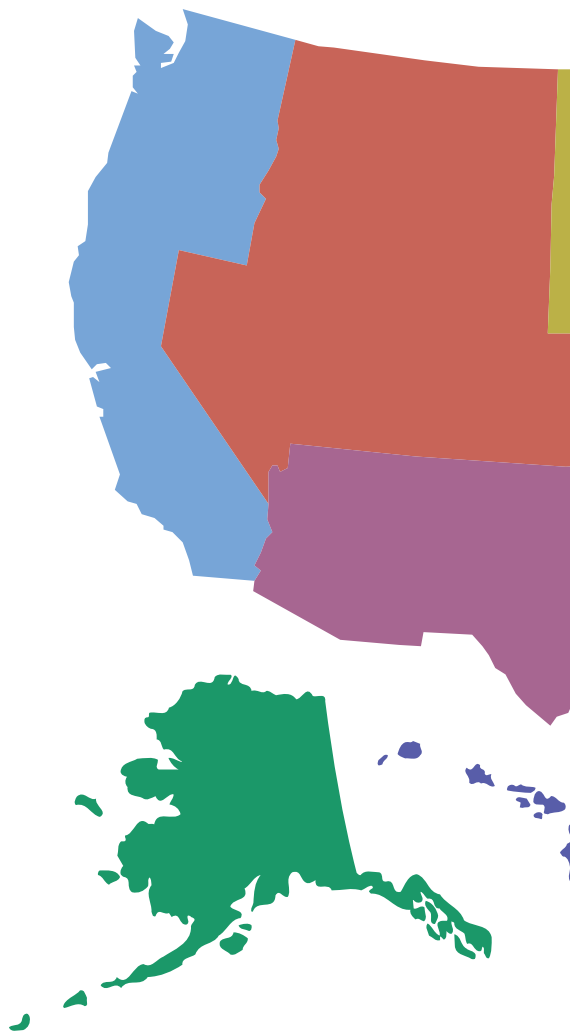
OFFER SUBJECT TO CHANGE WITHOUT NOTICE

ColdFusion

For more information go to...

U.S.

Alabama Huntsville Huntsville, AL CFUG www.nacflug.com	Colorado Denver Denver CFUG www.denvercfug.org/	Iowa Johnston Des Moines, IA CFUG www.hungrycow.com/cfug/
Alaska Anchorage Alaska Macromedia User Group www.akmmug.org	Delaware Kennett Square Wilmington CFUG www.bvcfug.org/	Kentucky Louisville Louisville, KY CFUG www.kymug.com/
Arizona Phoenix www.azcfug.org	Delaware Laurel Delmarva CFUG www.delmarva-cfug.org	Louisiana Lafayette Lafayette, LA MMUG www.cflib.org/acadiana/
Arizona Tucson www.tucsoncfug.org	Florida Jacksonville Jacksonville, FL CFUG www.jaxfusion.org/	Maryland Lexington Park California, MD CFUG http://www.smdcfug.org
California San Francisco Bay Area CFUG www.bacflug.net	Florida Winter Springs Gainesville, FL CFUG www.gisfusion.com/	Maryland Rockville Maryland CFUG www.cfug-md.org
California Riverside Inland Empire CFUG www.sccfug.org	Florida Plantation South Florida CFUG www.cfug-sfl.org	Massachusetts Quincy Boston, MA CFUG www.bostoncfug.com
California EL Segundo Los Angeles CFUG www.sccfug.org	Florida Tallahassee Tallahassee, FL CFUG www.tcfug.com/	Michigan East Lansing Mid Michigan CFUG www.coldfusion.org/pages/index.cfm
California Irvine Orange County CFUG www.sccfug.org	Florida Palm Harbor Tampa, FL CFUG www.tbmmug.org	Minnesota Brooklyn Park Twin Cities CFUG www.colderfusion.com
California Davis Sacramento, CA CFUG www.saccfug.org	Georgia Atlanta Atlanta, GA CFUG www.acfug.org	Missouri Overland Park Kansas City, MO CFUG www.kcfusion.org
California San Jose (temporary) Silicon Valley CFUG www.siliconvalleycfug.com	Illinois East Central East Central Illinois CFUG www.ecicfug.org/	Missouri O'Fallon St. Louis, MO CFUG www.stlmmug.com/
California San Diego San Diego, CA CFUG www.sdcfug.org/	Indiana Avon Indianapolis, IN CFUG www.hoosierfusion.com	New Jersey Princeton Central New Jersey CFUG http://www.cjcfug.us/
California Long Beach Southern California CFUG www.sccfug.org	Indiana Mishawaka Northern Indiana CFUG www.ninmug.org	Nevada Las Vegas Las Vegas CFUG www.snfcug.com/



New York Albany Albany, NY CFUG www.anycfug.org	North Carolina Raleigh Raleigh, NC CFUG www.ccfug.org
New York Brooklyn New York, NY CFUG www.nycfug.org	Ohio Dayton Greater Dayton CFUG www.cfd Dayton.com
New York Syracuse Syracuse, NY CFUG www.cfugcny.org	Oregon Portland Portland, OR CFUG www.pdxcfug.org

User Groups

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



Pennsylvania
Carlisle
Central Penn CFUG
www.centralpenncfug.org

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

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

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

Tennessee
LaVergne
Nashville, TN CFUG
www.ncfug.com

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

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

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

Texas
Houston
Houston Area CFUG
www.houcfug.org

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

Washington
Seattle
Seattle CFUG
www.seattlecfug.com

About CFUGs

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

INTERNATIONAL



Australia
ACT CFUG
www.actcfug.com

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

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

Australia
Victoria CFUG
www.cfcentral.com.au

Australia
Western Australia CFUG
www.cfugwa.com/

Italy
Italy CFUG
www.cfmentor.com

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

Scotland
Scottish CFUG
www.scottishcfug.com

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

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

Brazil
Brasilia CFUG
www.cfugdf.com.br

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

Brazil
Sao Paulo CFUG
www.cfugsp.com.br

Canada
Kingston, ON CFUG
www.kcfug.org

Canada
Toronto, ON CFUG
www.cfugtoronto.org

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

Spain
Spanish CFUG
www.cfugspain.org

Switzerland
Swiss CFUG
www.swisscfug.org

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

Turkey
Turkey CFUG
www.cfrtr.net

United Kingdom
UK CFUG
www.ukcfug.org



Easy Flash Dashboards

Using CFMX-delivered dynamic data



By Tim Burton

Dashboards are user interfaces that organize and present information intuitively,

usually with graphical elements. The information is often aggregated from several sources (databases, operational metrics,

log files, etc.) and summarized. They are often used by decision or policy makers.

Using CFMX v7's <cfform...> tag, you can build Flash-based forms whose "controls" look much like those found on standard "fat clients"; you can also build charts and graphs. However, combining them in a typical dashboard application results in a rather hodgepodge look-and-feel. Of course, you can hand-code Flash reports and create dial-like controls and eye-catching animations if you have the necessary expertise and time, but if you lack either, you might want to take a look at Crystal Xcelsius (CX). This article briefly describes the product and shows how to deploy very slick-looking, interactive, Flash-driven dashboards using CFMX-delivered dynamic data. It assumes you've installed the CX development software, the IDE.

As it's common practice to import corporate data into spreadsheets for presentation and decision-making purposes, CX's functionality is based upon Excel. Therefore, to build a CX dashboard you must create a spreadsheet containing the data tables (defined rows and columns) and cells that will ultimately be transformed into graphical "eye candy." After performing this step and saving the spreadsheet, the CX IDE is opened, a new project created, and the spreadsheet imported into the project; the latter can be thought of as embedded data structures. You now have the data; the next step is to construct charts, dials, sliders, and other components. Wizards then help you map the data to the component(s). The last step within the CX IDE compiles the dashboard into a Flash file (with suffix

.swf), which can be sent to users via e-mail or deployed to a Web server for browser viewing.

Let's look at an example.

Figure 1 shows one tab of a dashboard containing six Flash components. Starting from the top right and viewing clockwise, notice:

1. The multi-colored chart showing the output from each of several different service groups for each month of 2005
2. A button titled "Chart Selected Year" (a specific instance of a component called an "XML Data Button")
3. A combo box for choosing a year
4. A list box for choosing the type of output to be shown (in this case, either the number of contracts or the total \$ contract amount for each service group)
5. A dial showing "Contracting Workload"
6. An alert showing the workload trend

The data shown on this, or any dashboard for that matter, can be static or it can be dynamic. If static, it means the

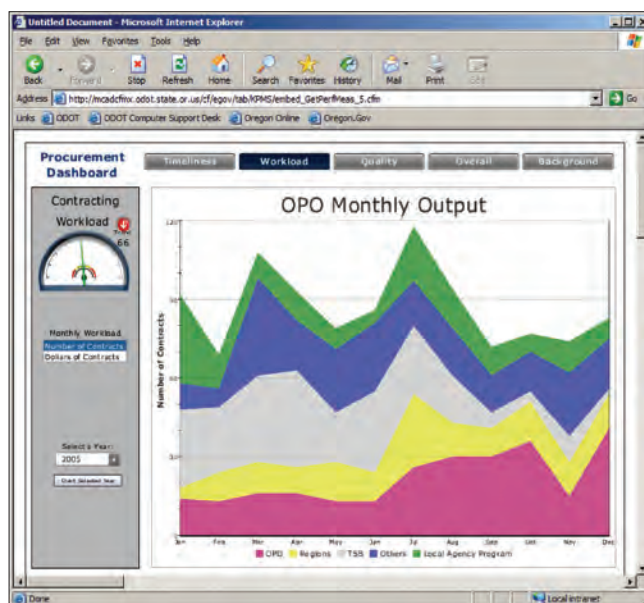


Figure 1

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 \$29.89 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



Reach Over 100,000 Enterprise Development Managers & Decision Makers with...



Offering leading software, services, and hardware vendors an opportunity to speak to over 100,000 purchasing decision makers about their products, the enterprise IT marketplace, and emerging trends critical to developers, programmers, and IT management

Don't Miss Your Opportunity
to Be a Part of the Next Issue!

Get Listed as a Top 20* Solutions Provider

**For Advertising Details
Call 201 802-3021 Today!**

*ONLY 20 ADVERTISERS WILL BE DISPLAYED. FIRST COME FIRST SERVE.



The World's Leading i-Technology Publisher



flash

data resides in the spreadsheet embedded within the compiled Flash file. If dynamic, it means the data would be retrieved from, say, a back-end database. This is made possible by the XML Data Button component, which sends data from the Flash executable to a back-end CF application, receives the response, repopulates the embedded spreadsheet's tables and cells, and refreshes the visual components. This essentially gives you Flash remoting without a lot of hassle.

In our example, the input combo box is configured to show a range of years that is mapped to a column of spreadsheet cells containing those year-values; its output (the year chosen by the user) is mapped to one of those spreadsheet cells. When the user hits the "Chart Selected Year" button, the value in that one cell is wrapped into an XML packet and sent to a ColdFusion action template that has been designated by a URL in the wizard. At the CF server, the XML packet is parsed and the value (the year in this case) is passed into a database query; data from a static file could, of course, be returned as well. The CF server then wraps the query resultset into an XML packet for return to the Flash file and subsequent display (see Listing 1).

The XC IDE generates the entire structure of the XML packet for sending to and receiving from the CF server, thus simplifying the job of parsing the XML in code. Figure 1 shows four sections of CF code handling these back-end tasks.

Section 1: The data passed in from the .swf is converted (by the CF server) into a struct with an extraneous name=value pair ("FIELDNAMES"), which is removed in this section. What remains is the #form# variable with only the original XML packet created by the .swf file.

Section 2: To prevent the XML parser from throwing an error, the entire #form# variable is changed to lower case. Next, an XML tree object is created and the data (in this case, the year) is extracted.

Section 3: The database is queried by a stored procedure call with the year passed in as the input parameter.

Section 4: The returned cfquery is converted into an XML string for delivery back to the Flash file.



Workflow

We've developed enough dashboards to recognize that it's an iterative process, usually involving an XC "power user," a business expert who knows what data should be displayed,

BY NOW THERE ISN'T A
SOFTWARE DEVELOPER ON EARTH
WHO ISN'T AWARE OF THE
**COLLECTION OF PROGRAMMING
TECHNOLOGIES KNOWN AS AJAX!**

REAL - WORLD AJAX

ONE DAY SEMINAR

www.ajaxseminar.com

March 13, 2006

Marriott

Marriott Marquis Times Square
New York City

For more information
Call 201-802-3022 or
email events@sys-con.com

REAL WORLD

How, in concrete terms, can you take advantage in your own projects of this newly popular way of delivering online content to users without reloading an entire page?

How soon can you be monetizing AJAX?

This "Real-World AJAX" one-day seminar aims to answer these exact questions...

Led by "The Father of AJAX" himself, the charismatic Jesse James Garrett of Adaptive Path, "Real-World AJAX" has one overriding organizing principle: its aim is to make sure that delegates fortunate enough to secure themselves a place – before all seats are sold out – will leave the seminar with a sufficient grasp of Asynchronous JavaScript and XML to enable them to build their first AJAX application on their own when they get back to their offices.

HURRY!
LIMITED SEATING
THIS SEMINAR WILL
SELL-OUT
CALL 201-802-3022
TO REGISTER!



Jeremy Geelan
Conference Chair, Real-World AJAX
jeremy@sys-con.com



Jesse James Garrett
Father of AJAX



Scott Dietzen
Creator of WebLogic, Ph.D., President and CTO, Zimbra



Bill Scott
AJAX Evangelist of Yahoo!



David Heinemeier Hansson
Creator of Ruby on Rails



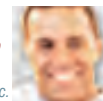
Satish Dharmaraj
Father of JSP, Co-Founder & CEO, Zimbra



Rob Gonda
Best-Selling AJAX Author, CTO, iChameleon Group



Dion Hinchliffe
Co-founder & CTO, Sphere of Influence Inc.



Ross Dargahi
Well-known AJAX Evangelist & Co-founder and VP of Engineering, Zimbra

Early Bird	\$995
<small>(Before January 31, 2006)</small>	
Discounted Price	\$1,195
<small>(Before February 28, 2006)</small>	
Seminar Price	\$1,295
<small>(After February 28, 2006, and if any seat is available)</small>	

MEDIA SPONSOR




LIVE SIMULCAST!
AROUND THE WORLD ON SYS-CON.TV

PRODUCED BY
sys-con
EVENTS

an IS analyst who knows the backend database, and a CF developer (see Figure 2).

Summary

The product can be seen as complementary to and as an extension of existing CFMX functionality. The Xcelsius/Flash deployables can, of course, do more than dashboards. Indeed, there are an impressive number of configurable components available within the XC IDE (see the XC Web site: http://www.xcelsius.com/Products/XL_products.html). The vendor also has a component that purports to offer dynamic messaging between the Flash executable and Web services; however, we couldn't get it

to work and, given that the equivalent XML data button serves our intranet purposes well, we didn't try to solve that problem. The documentation, unfortunately, is replete with .asp code and examples but not .cfm code. 

About the Author

Tim Burton is the eGovernment Applications Architect for a large state agency in Oregon and has been writing CFML since 1998. This is his third career; he previously practiced medicine and made art (metal sculpture).

tim.burton@odot.state.or.us

Figure 1 ---- CF action template for responding to Flash request

```
<cfprocessingdirective suppresswhitespace="yes">
<!-- SECTION 1 ----->
<cfset theParsedInput = "">
<cfloop collection="#form#" item="inputFormKey">
    <cfif inputFormKey neq "FIELDNAMES">
        <cfset theParsedInput = "#theParsedInput#" &
"#inputFormKey#=#form[inputFormKey]#">
    </cfif>
</cfloop>

<!-- SECTION 2 ----->
<cfset theParsedInput = "#LCase(ToString(theParsedInput))#">
<cfset theInputTree = #XMLParse(theParsedInput)#>
<cfset theYearIn = theInputTree.XmlRoot.XmlChildren[1]
    .Row.Column.XmlText>

<!-- SECTION 3 ----->
<cfstoredproc procedure="GetPerfs" datasource="#request.DSN#">
    <cfprocparam type="In"
        cfsqltype="CF_SQL_INTEGER"
        dbvarname="@theYear"
        value="#theYearIn#"
        null="no">
    <cfprocresult name="theRecords">
</cfstoredproc>

<!-- SECTION 4 ----->
<cfset sXML = "<data>" & "<variable name=" & "#chr(34)#" &
    "theResult" & "#chr(34)#" & ">">
```

```
<cfloop query="theRecords">
    <cfset sXML = sXML & "<row>">
    <cfset sXML = sXML & "<column>" & "#Trim(mth)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(jan)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(feb)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(mar)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(apr)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(may)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(jun)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(jul)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(aug)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(sep)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(oct)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(nov)#" & "</column>">
    <cfset sXML = sXML & "<column>" & "#Trim(dec)#" & "</column>">
    <cfset sXML = sXML & "</row>">
</cfloop>
<cfset sXML = sXML & "</variable>" & "</data>">

<!-- SECTION 5: the output ----->
<cfcontent type="text/plain">
<cfoutput>#sXML#</cfoutput>

</cfprocessingdirective>
```

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

BALANCE

Designer/developer, front-end/back-end, clients/sanity. . .web development is a balance and we can help you maintain it. Join now and experience a wealth of training resources tailored to the tools you use every day.

www.communitymx.com



Visit www.communitymx.com/trial/ for your free 10 day trial.



Intermedia.NET...

Now serving the hottest ColdFusion.

© Copyright INTERMEDIA.NET, Inc 2005. All rights reserved. All other trademarks are property of their respective holders.

At Intermedia.NET we go beyond the industry standard by supporting the hottest new Coldfusion software, offering power like never before. For nearly a decade, we've been providing reliable, secure hosting to thousands of companies across the globe. We can do the same for you.

Intermedia.NET's premier hosting services include:

- ColdFusion MX hosting
- Custom tag registration
- Competitive plans
- Verity collections search engine
- Security sandboxes
- Guaranteed service levels

Unprecedented power, unmatched reputation...

Intermedia.NET is your hosting solution.



INTERMEDIA.NET

Call us at: 1.888.379.7729

e-mail us at: sales@intermedia.NET

Visit us at: www.intermedia.NET