SYS-CON

TALK IS CHEAP... LET'S SEE SOME CODE PG 5

ColdFusion Developer's Journal

ColdFusionJournal.com

January 2006 Volume:8 Issue:1

Presorted
Standard
US Postage
PAID
St. Croix Press

PLUS...

macromedia®

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.

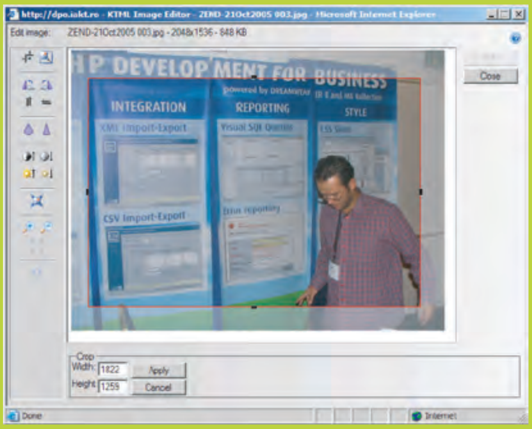
To learn more or take ColdFusion MX 7 for a test drive, go to:
macromedia.com/go/cfmx7_demo

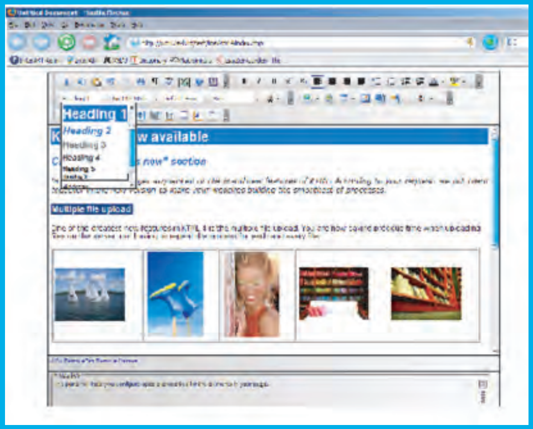
KTML4 Word editing in browser

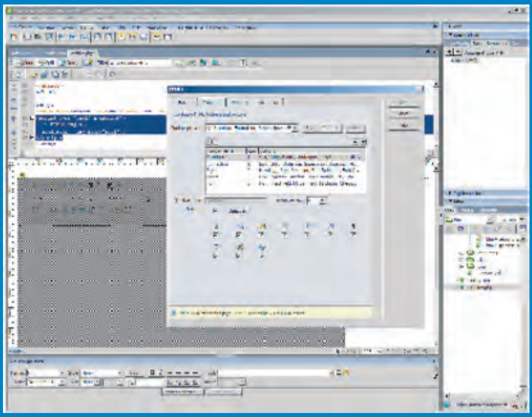
Along with the most affordable **UNLIMITED** licence

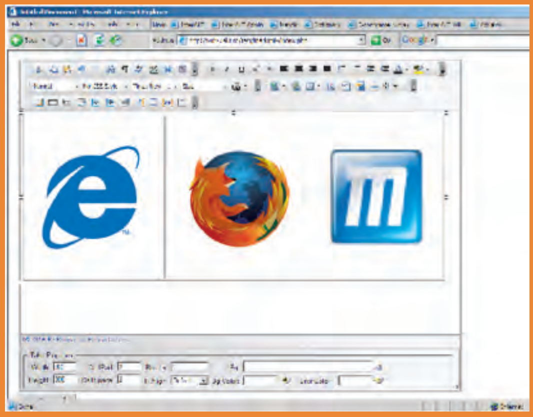
Revolutionary Image Editor

Word-like visual CSS styles


http://dpo.iakt.ro - KTML Image Editor - ZEND-21Oct2005 003.jpg - Microsoft Internet Explorer
Edit image: ZEND-21Oct2005 003.jpg - 2048x1536 - 848 KB
Close
Crop
Width: 1822
Height: 1259
Apply
Cancel
Done
Internet







Fast Dreamweaver integration

Wide browser compatibility

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

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

CFDynamics

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.

macromedia ALLIANCE PARTNER

Paper|Thin Partner

BlueDragonCertified V.I.P. Hosting Partner

Visit us online or call to order!

WWW.CFDYNAMICS.COM

1 - 866 - 233 - 9626
866 - CFDYNAMICS

COLDFUSION Developer's Journal

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
Bahadir Karuv, PhD bahadir@sys-con.com

production

production consultant
Jim Morgan jim@sys-con.com

lead designer
Abraham Addo abraham@sys-con.com

art director
Alex Botero alex@sys-con.com

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

contributors to this issue
Isaac Dealey, Ben Edwards, Hal Helms, Simon Horwith, Jeff Peters, David Ross, Neil Ross, Chris Scott, Nicholas Tunney, Matt Woodward

editorial offices

SYS-CON MEDIA
135 Chestnut Ridge Rd., Montvale, NJ 07645
Telephone: 201 802-3000 Fax: 201 782-9638
COLDFUSION DEVELOPER'S JOURNAL (ISSN #1523-9101)
is published monthly (12 times a year)
by SYS-CON Publications, Inc.

postmaster: send address changes to:
COLDFUSION DEVELOPER'S JOURNAL
SYS-CON MEDIA
135 Chestnut Ridge Rd., Montvale, NJ 07645

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

Talk Is Cheap. . . Let's See Some Code

By Simon Horwith

I've been a certified ColdFusion instructor for six years and speak regularly at user groups and conferences. One thing I've learned without a shadow of a doubt is that talk is cheap. Anyone can talk a good talk, but the proof is always in the pudding. Okay, enough clichés.

My point is that developers learn best from doing things themselves and from seeing other developers write the code. To get the most from the articles featured in *ColdFusion Developer's Journal*, you have to try the techniques or download the code and really gain an understanding of how it works. This issue is definitely no exception to that rule.

This month we take a look at frameworks. There has been a lot of talk in the community about frameworks, and new ones are springing up seemingly every day. How is a developer to keep up? Surely, what the community needs are resources that allow them to effectively compare and contrast the various frameworks on an even playing field. So this month I decided to try something new: I asked a select group of pioneers and experts with the popular frameworks to build an application with their framework and write an article about it. To even the playing field, I've asked all of the developers to develop the exact same application. Each of our articles this month is a case study on how the Macromedia CF Pet Market application would be implemented using a specific framework. The original application was developed by Mike Nimer at Macromedia shortly after ColdFusion MX was released. I suggest you take a look at the original application using the following URLs:

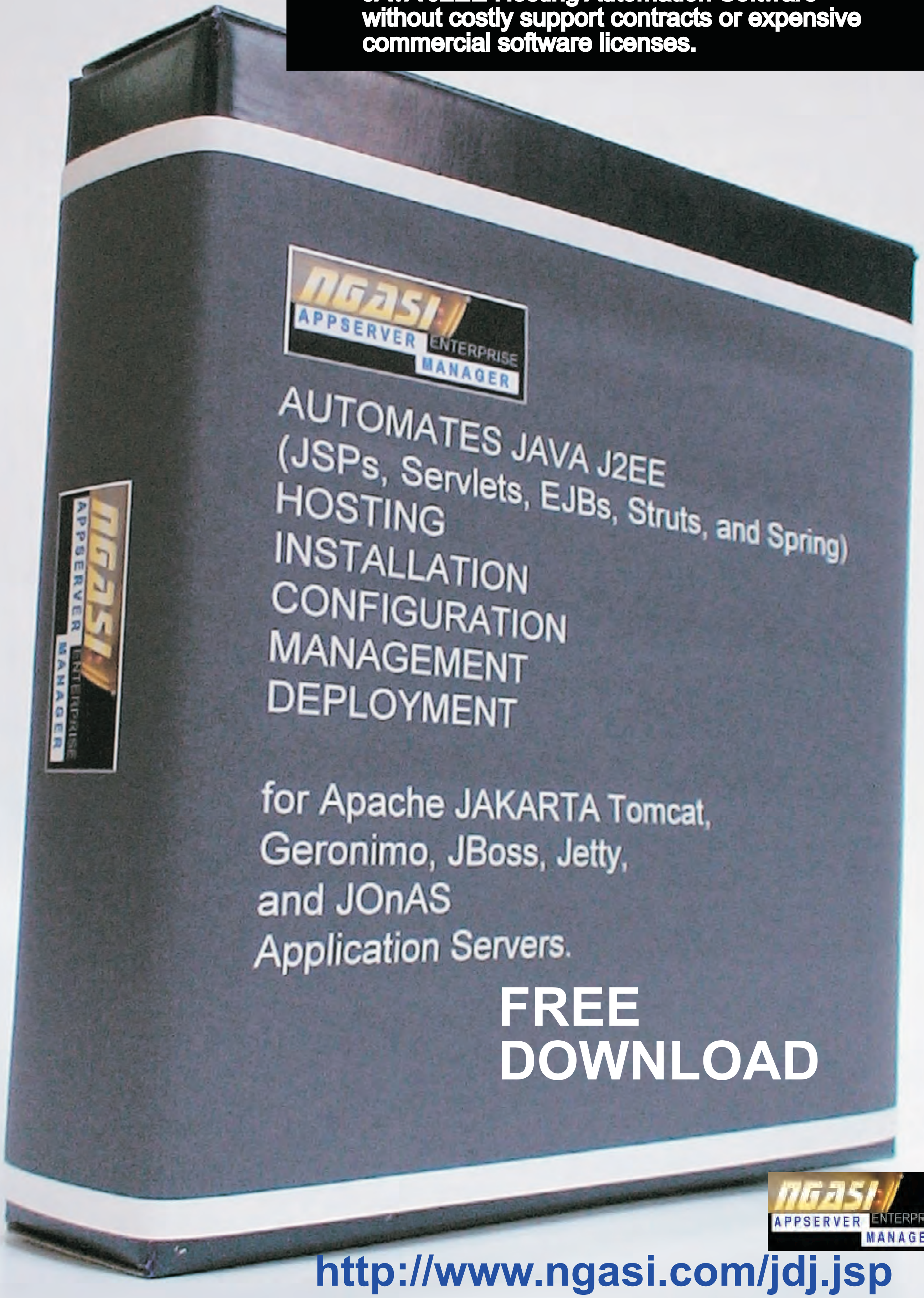
- ***Files:*** http://www.macromedia.com/bin/petmarket.cgi (be sure to select the radio button for the ColdFusion MX [CFML/HTML] version).
- ***FAQ:*** http://www.macromedia.com/devnet/blueprint/articles/faq.html
- ***General Info:*** http://www.macromedia.com/devnet/coldfusion/articles/petmarket.html

As I said, new framework and new versions of existing frameworks are being released at, dare I say it, an alarming rate.

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

JAVA J2EE Hosting Automation Software without costly support contracts or expensive commercial software licenses.


NGASI APPSERVER ENTERPRISE MANAGER
AUTOMATES JAVA J2EE
(JSPs, Servlets, EJBs, Struts, and Spring)
HOSTING
INSTALLATION
CONFIGURATION
MANAGEMENT
DEPLOYMENT
for Apache JAKARTA Tomcat, Geronimo, JBoss, Jetty, and JOnAS Application Servers.

FREE DOWNLOAD

NGASI APPSERVER ENTERPRISE MANAGER

http://www.ngasi.com/jdj.jsp

1.866.256.7973

If it's difficult for developers to keep up, it's even more difficult for a magazine. This issue is just the beginning. In addition, along with the folks at AboutWeb, I am making another new resource available: www.cfpetmarket.com. Cfpetmarket.com is a site that allows developers to search for and download different versions of the CF Pet Market application. It also allows developers to upload their own version of CF Pet Market. As new frameworks are released, and new versions of frameworks released, it is my hope that developers will continue to upload more versions of CF Pet Market to the site to serve as examples to the community. The site is not limited to frameworks; anyone who wants to upload a version of CF Pet Market is welcome to upload their idea of the ideal way to develop this application. All of the sample applications from the articles in this issue are available for download now at http://www.cfpetmarket.com. You can search for them by framework, methodology, and/or author name.

All of this month's authors were given the following restrictions: each application must have the exact same look and feel of the original and the code must run off of the original database structure. The goal is not to change the original business requirements or user interface but to offer a new version of the code. These same requirements apply to new submissions to the cfpetmarket.com site as well.

The frameworks/methodologies and authors featured in this issue are:

- ***Fusebox:*** Jeff Peters
- ***MACH II:*** Hal Helms and Ben Edwards
- ***onTap:*** Isaac Dealey
- ***TheHUB:*** Neil Ross
- ***Model Glue:*** Nicholas Tunney
- ***ColdSpring:*** David Ross and Chris Scott
- ***SAM:*** Simon Horwith

Hopefully, you see some frameworks or methodologies on that list that you've either never heard of or have been meaning to learn more about. That's what the cfpetmarket.com project and this issue are all about: learning alternative, even better, methods for developing applications.

It's been a year now since I began running "deep focus" issues of *CFDJ* and the response from our readers has been overwhelmingly positive. Unfortunately, filling an issue with articles about a single topic is a lot more difficult than you might think. The editorial calendar for 2006 is now available online at http://res.sys-con.com/section/5/CFDJEdCal2006.pdf. I encourage any of you who are interested in contributing to making *CFDJ* the best resource on the planet for ColdFusion developers to take a look at the calendar and e-mail me at simon@horwith.com if any of the topics strike your fancy. If you think that filling an issue with articles about one topic is difficult, then you can only imagine how much work it was to fill an issue with articles and authors as specifically focused as we have this month. Several months ago I set out to produce the best (or most useful, at least) issue in ***ColdFusion Developer's Journal*** history and to kick off the new year by offering it and the cfpetmarket.com site to the community. This one has been a labor of love for me. I hope that you enjoy reading it as much as I've enjoyed putting it together.

"Each of our articles this month is a case study on how the Macromedia CF Pet Market application would be implemented using a specific framework"

SYS-CON MEDIA

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

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

group publisher
Jeremy Geelan jeremy@sys-con.com

advertising

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

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

advertising director
Robyn Forma robyn@sys-con.com

advertising manager
Megan Mussa megan@sys-con.com

sales & marketing director
Dennis Leavey dennis@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 designer
Stephen Kilmurray stephen@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

Pet Market Remix: TheHUB

A comparison

By Neil Ross

I recently spoke to a group of attendees at the Fusebox and Frameworks Conference 2005 about the framework that I call TheHUB. Then when I was asked to write this article, I thought it would be a great way to compare and contrast it with several of the frameworks discussed at that conference. So I jumped at the chance to share my approach to developing ColdFusion applications.

I use a framework and methodology called TheHUB, which is an approach that I developed over the past couple of years, and I discussed the origins and motivations for this approach at the conference. The bottom line is that TheHUB is easy to learn, easy to use, and easy to maintain. It has a much smaller footprint than other application frameworks and is flexible enough to allow you to develop applications the way you want without adding lots of restrictions to the way you code.

The Basics

The first thing to understand about TheHUB is that all you're being asked to do is to conform to a pattern of application development. This pattern is similar to the development of a Fusebox application where you indicate a "fuseaction" for each request.

When following the pattern of development prescribed by TheHUB, you format your hyperlinks and form actions in such a way that you identify the template to be loaded by passing along the URL, a key/value pair that specifies a template to be loaded from a specific directory within the folder structure of your site or application. The analysis and processing of the request is implicit and there is a piece of core code that TheHUB uses to analyze the query string on your request and load the appropriate templates.

This approach leaves you, the developer, free to decide how to organize the code, for example, along the lines of what is display code and what is not or along the subject area line. This gives you flexibility as the application evolves and expands.

Preparing the ColdFusion Server

The "readme" file that comes with the downloadable Pet Market code includes the server setup information. You'll need to create your data source connection within the ColdFusion administrator. You'll also need to create a virtual directory in IIS or a virtual mapping in the built-in Web server. Do this by following the directions in the "readme" file included with the download. You'll edit the "jrun-web.xml" file to include the mapping to your Pet Market directory.

Preparing TheHUB

TheHUB has very few core files that are needed for the actual operation of an application. I actually removed several of the sample files that come with the download because I didn't need them for the application to function (see cfpetmarket.com). Now, when I do a file count and comparison I find that there were 100 files in the original application and only 102 in the converted application.

There are a few lines of code in the core files that you'll need to update. TheHUB relies on a variable called "request.frameworkhub" in order for an application to function properly. This variable is set within the "globals.cfm" within the "cmn" directory of the core files download for TheHUB and points to the page that you want to run all of the page requests through. The code within the "index.cfm' is pretty simple and just handles the include of the requested templates:

```
<cfinclude template="cmn/varhandler.cfm">
<cftry>
   <cfinclude template="#request.dir##request.tmp#.cfm">
<cfcatch type="missinginclude">
   <script>
         alert("The page that you requested has experienced an error.
You'll now be redirected back to the page you last viewed.");
         history.back();
   </script>
</cfcatch>
```

```
</cftry>
```

This is normally the "index.cfm" template, but it can be another template as long as you copy the code over to that new template and update the reference to it in the "globals.cfm".

For this application, update the "cmn\globals.cfm" so that all you need in there is:

```
<cfscript>
    request.frameworkhub="/petmarket/index.
cfm?";
request.cfcPath="petmarket.extensions.compo-
nents.petmarket";
</cfscript>
```

Normally there are other variables set within this block, but they're not needed because they are a part of the Pet Market application that we are converting.

Organizing the Code

I'm not going to pretend to know the best way for you to organize code for any application that you're working on today or tomorrow. I know that every developer thinks about development and how he or she wants the applications structured, so TheHUB is nice enough to let you organize your code so that it makes sense to you. I got started converting the Pet Market code by doing a quick review of the application. After looking through the application for a few minutes, I decided to organize the code into three main groupings or subject areas:

1. ***DSP:*** Generic display code
2. ***Cart:*** Display code specific to cart functionality
3. ***Extensions:*** Intact structure from the downloaded zip

I created a directory for my application and called it "PetMarket" so that I could run the applications side-by-side. I wanted to make sure that the functionality was the same from screen to screen and that my interaction with the application didn't differ with any action that I took within the application.

Updating the Code

I mentioned earlier that the most important thing to know about TheHUB is the format of the hyperlinks and form actions. The next step to converting the Pet Market application to a "TheHUB" application is to reformat all of the hyperlinks and form actions. All requests will flow through the "frameworkhub". This is just the template used to conduct traffic for the application. You'll need to update all hyperlinks and form actions to flow through that template.

Let's look at an example. If the hyperlink you're updating goes to the "preferences.cfm" template in the directory where you located that template, in my case I put it in the "dsp" directory.

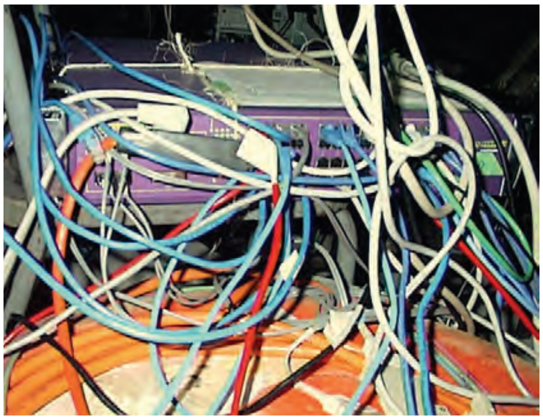
Before:

```
<a href="preferences.cfm">preferences</a>
```

After:

```
<a href="#request.frameworkhub#dsp=preferences"
>preferences</a>
```

The key to making these changes is consistency. I ran a "Find All" for both "<a href=" and for "action=" so that I could modify each form action and all hyperlinks within the application. Everything functions the same but I'm running all requests through TheHUB now.



Beyond the Basics

In addition to the changes to the code for the hyperlinks and form actions, there were a couple of other small changes that I made:

1. Update CFLOCATION tags to use the "request.frameworkhub" variable
2. Update CFC object instantiation calls to use the "request.cfcpath" variable

That's about it. The longest part of the process was reviewing the code from the original Pet Market download files. Of course, not every application will be as easy to convert as this one.

Conclusion

This exercise was undertaken so that you would have a side-by-side comparison of the different application frameworks in action. It should give you a good idea of the complexity of the frameworks highlighted in these versions of the Pet Market application. This article shows that the work involved in moving your development approach to TheHUB is actually pretty painless. It gives you all of the flexibility that you need to make good decisions about code organization, maintenance tasks, and the extension of the application. Also note that if you compare them, TheHUB has a much smaller footprint than other frameworks and does not clutter up your application folders with files that you'll never use or even understand.

If you start from scratch using TheHUB as your application framework, you'll find that you can easily extend and maintain the applications that you're building without the overhead of maintaining configuration files.

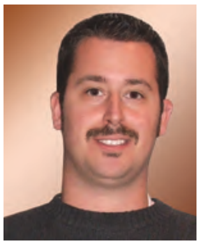
About the Author

Neil Ross is the system development manager for the Pennsylvania Office of Attorney General and owner of Codesweeper, an application and Web site development firm specializing in ColdFusion development and consulting. He is a Certified ColdFusion developer and was a ColdFusion instructor during his days with Allaire Corporation. Neil is a frequent speaker at CFUG and MMUG meetings as well as conferences like MAX, CFUN, and CF/MXEurope and an author of articles and co-author of Inside ColdFusion MX.

neil@codesweeper.com

The ColdFusion Pet Market Blueprint Application in Model-Glue

Separating business logic from the presentation layer



By Nicholas Tunney

After unzipping the Blueprint Application and configuring my ColdFusion instance to get the application running as-is, I poked around the code to see how it was laid out. It was pretty straightforward: create an object of the component we need to access, call a method in the object, and display the results. This is pretty standard in most applications. Model-Glue (M-G) can easily handle this kind of architecture even though it's not in an MVC pattern.

That's a great strength of Model-Glue. While it facilitates MVC, your application doesn't have to use this pattern to use M-G. Since the task presented to me was to alter the Blueprint Application to use Model-Glue, I decided at this point to keep as much of the original architecture as possible and not re-architect the application in an MVC pattern.

To get started, I downloaded Joe Rinehart's Model-Glue from http://www.model-glue.com. I put the Model-Glue folder in my webroot, and mapped /modelglue in the ColdFusion Administrator. In the download there's a directory called Model GlueApplicationTemplate. This folder should be copied to a new location and renamed to match your new application. This folder contains default message-listeners, event-handlers, etc. that Model-Glue will use during runtime for this specific application.

Before I get into the Pet Market application, let me explain exactly what Model-Glue is and how it works. Developed by Joe Rinehart, Model-Glue was created as a framework to help developers separate business logic from the presentation layer. Model-Glue uses a simple XML schema so the developer can define event-handlers and message-listeners. A visual representation of how M-G works is shown in Figure 1.

When a request is placed, an event is passed to the framework via a URL or FORM variable. In Model-Glue, FORM and URL variables end up in one Event object. From your view, you access this object as viewState. In a controller, you access this object as Event. The framework automatically pushes this event object to each controller as an argument (arguments.event). Now, as I said above, I could easily use M-G and architect this application using MVC. Each controller attached to the message-listeners would simply have to accept the Event argument. Anyone who's read up on MVC knows that controllers should contain no interaction with the data persistence layer. Only the Model should interact with your data persistence. Figure 2 shows how Model-Glue fits into the MVC pattern.

In the case of the Pet Market Blueprint Application, we could either use the existing architecture and have Model-Glue act as an intermediary and just call the appropriate view templates that, in turn, access the Model directly, or rewrite the application so Model-Glue calls the controller and returns the data to the view as an MVC patterned architecture should. I think that a full re-architecture is outside the scope of this article and will first get the application running using M-G, and second, re-architect one template to use M-G in an MVC pattern as an example that better shows the power of M-G.

Now that you have a basic understanding of Model-Glue and our plan of attack for this application, let's delve into the Pet Market Blueprint Application and set it up to use Model-Glue.

Before we begin converting our application, we have to determine where to put our files. I generally like to have an assets directory where I keep all items consumed by the application. This doesn't include display templates (view), but does include things like images, components that belong to my application's Model, stylesheets, etc. Model-Glue will work with any directory structure. I just like to keep my directory layout consistent across my applications so it's easy to determine where the items I need are located. Notice that the assets folder only contains components related to our Model. I've learned to keep all controller components in a controller folder under the application root. Once again, this is personal preference and M-G doesn't require any specific directory structure. As a framework, it's very forgiving.

We also need a place to hold our view templates. When using Model-Glue, I usually create a folder in the application root called view. Under the view directory, we can create folders to hold the views specific to our separate modules. While reviewing the existing Blueprint Application code, it seemed like the application views broke into four distinct modules: Search, Products, Checkout, and static templates that would fall into a more general group. In the root of view we can store our general view templates. These templates would include files such as about.cfm, legal.cfm, and other templates that fall outside our application's general functionality. I then created a directory for each of the other modules above (Search, Product, and Checkout).

I now copy over the existing files in the Pet Market Blueprint Application into the appropriate folders. When I'm done, my directory structure looks like Figure 3.

To configure our new Model-Glue application, we edit config/

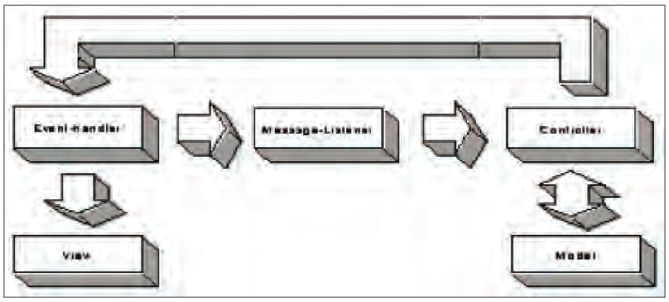


Figure 1: How Model-Glue works

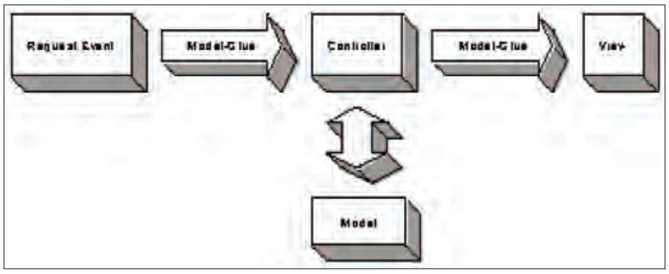
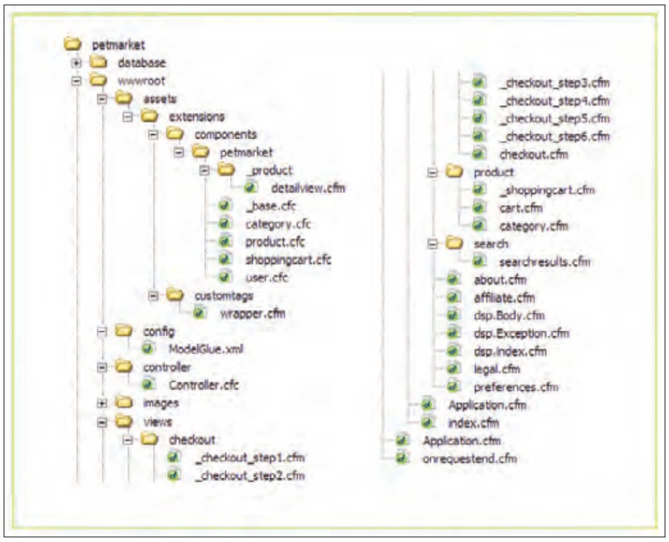


Figure 2: How Model-Glue fits into the MVC pattern



```
petmarket
  database
  wwwroot
    assets
      extensions
        components
          petmarket
            _product
              detailview.cfm
            _base.cfc
            category.cfc
            product.cfc
            shoppingcart.cfc
            user.cfc
        customtags
          wrapper.cfm
    config
      ModelGlue.xml
    controller
      Controller.cfc
    images
    views
      checkout
        _checkout_step1.cfm
        _checkout_step2.cfm
        _checkout_step3.cfm
        _checkout_step4.cfm
        _checkout_step5.cfm
        _checkout_step6.cfm
        checkout.cfm
      product
        _shoppingcart.cfm
        cart.cfm
        category.cfm
      search
        searchresults.cfm
      about.cfm
      affiliate.cfm
      dsp.Body.cfm
      dsp.Exception.cfm
      dsp.index.cfm
      legal.cfm
      preferences.cfm
    Application.cfm
    index.cfm
  Application.cfm
  onrequestend.cfm
```

Figure 3: Directory structure

Modelglue.xml. The Modelglue.xml file is broken into three distinct sections: config, controllers, and event-handlers. <Config> contains our application specific configuration settings. <Controllers> defines the location of our controller files and the message-listeners attached to each controller. Every broadcast defined in our event-handler will have one message-listener defined pointing to a method in a controller (refer back to Figure 1). The last section, <event-handlers>, listens for events broadcast during our request. The first event-handler we'll intercept is the event passed from our user (via the URL or a form). Each event handler can then broadcast its own messages that our controller message-listeners "hear" and call methods in our controllers. These controller methods interact with the Model (data persistence) and apply business logic. They then set data into the Event object and set an optional result. If a result's been set, it's interpreted by our event-handler and then the next message is broadcast or another event is fired or the view is displayed. Model-Glue is awesome in that it lets not only several methods be applied during a single request, but that the results set via our controllers can be interpreted by the event handler and another event can be broadcast, or one of several views can be called. Allowing the developer to use this conditional logic is unlike many other frameworks, and is very powerful.

When using Model-Glue, all requests are processed using the index.cfm page by default much like other frameworks. When a request is made from within your application, whether it be from a form action or a URL, it will have the format index.cfm?event=event_name. (Note: the event may also be passed as a form variable.) Event is the default variable name; however, you may customize M-G to listen for any variable name.

I now searched the wrapper.cfm file for all form actions and href links. I needed to have the request call index.cfm with the new event that I defined for each event-handler in ModelGlue.xml (see Listing 1 for an example event-handler). There were also two forms that onChange called submit() and posted back to the current page. I added hidden form fields named event and specified the event that it should post to. Since M-G looks in the Event Object for the event being called and URL and FORM variables are both compiled into this object, this still let me get the desired result. Model-Glue lets you specify which scope takes precedence (statePrecedence) in the <config> section of ModelGlue.xml (FORM or URL) in case both the FORM and URL scopes contain the same variable name.

Since I moved all the templates around and created a new directory structure, I had to point each template to the new location of the wrapper.cfm file. This was easy using a site-wide search and replace. Once the application was able to <cfimport> the wrapper, the application was up and running great, using the Model-Glue framework.

As promised, I then created a search controller to show how Model-Glue processes requests using MVC. The first thing to do is to set up ModelGlue.xml to broadcast a message to be intercepted by a controller, in this case SearchController.cfm (see Listing 2). In the sample application structure we copied over from Model-Glue, there's a template Controller you can use as an example. It includes three required methods by default. The init() method is called as a constructor. OnRequestStart() is invoked when the request first starts processing, and onRequestEnd() is the last method called for that request. I added a function called getSearchResults() to call our product object method search() and then set the results into the Event (see Listing 3). These results are now available in our viewState (see Listing 4). To see the complete controller and ModelGlue.xml, visit http://www.cfpetmarket.com and download the full code archive for this project.

One more thing I'd like to mention is the built-in Model-Glue debugging feature (see Figure 5). In the <config> section of ModelGlue.xml, there's a debug setting. When set to true, M-G caches important debugging information as your request is processed. This debug information is displayed at the bottom of your request (before the ColdFusion debugging information if you have debugging turned on in CFAdmin) regardless of whether or not an Exception is thrown. It shows information such as the time a specific action, such as an event-handler or message-listener, took to complete. The debug also displays results set in your controllers. In my opinion, the best debugging feature is the built-in trace method. In my controller I can trace any variable of any type by using the format arguments.event.trace("trace_name", variable). This traced value will then display in the Model-Glue debugging output. What a great feature to have at your disposal.

To close out this brief case study, let me say that integrating the Pet Market Blueprint Application into the Model-Glue framework took a total of 30 minutes. It was very simple and while this application didn't use many of M-G's powerful features, it clearly demonstrates that Model-Glue can handle applications of any size or complexity. Just how complex? I'm presently the team lead overseeing what is described as the largest production implementation of Model-Glue, and Model-Glue easily handles the load, reduced the learning curve for new team members learning MVC and OOP, and enabled us to keep the application code tidy. Thanks, Joe!

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

```
<!--
Name:   showPreferences
Module: General
Access: Public
Desc:   I display preference information.
-->
<event-handler name="showPreferences">
<broadcasts />
<views>
<include name="body" template="preferences.cfm" />
</views>
<results>
<result do="ShowTemplate" />
</results>
</event-handler>
```

Listing 2

```
<controllers>
<controller name="SearchController" type="petmarket.controller.Search-
Controller">
    <message-listener message="OnRequestStart" function="OnRequestStart"
/>
    <message-listener message="OnRequestEnd" function="OnRequestEnd" />
    <message-listener message="doGetSearchResults" function="getSearchRe
sults" />
  </controller>
</controllers>

<event-handlers>
<!-- search event handlers -->
<!--
    Name:   showSearchResults
    Module: Search
    Access: Public
    Desc:   I display search results.
  -->
  <event-handler name="search.showSearchResults">
    <broadcasts>
      <message name="doGetSearchResults" />
    </broadcasts>
    <views>
      <include name="body" template="search/searchresults.cfm" />
    </views>
    <results>
      <result do="ShowTemplate" />
    </results>
  </event-handler>
  …other event handlers…
</event-handlers>
```

Listing 3

```
<!--- getSearchResults() --->
  <cffunction name="getSearchResults" access="Public" returntype="void"
output="false" hint="I process a user search.">
    <cfargument name="event" type="ModelGlue.Core.Event"
required="true">

    <cfset var searchResults = variables.productObj.search(keyword=url.
keyword) />

    <cfset arguments.event.setValue("searchResults", searchResults) />
  </cffunction>
```

Listing 4

```
<cfscript>
  //search for all products

  searchResults = viewState.getValue('searchResults');
  …other code…
</cfscript>.
```

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

WEB DEV GURU seeks an Integrated Software Suite that speaks my language (XHTML) and won't cramp my style. Must play well with XML, CSS and others. I'm not superficial, but I like a nice code view. Clutter isn't cute.

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 web design and development tools you need, visit www.macromedia.com/go/8_studio8.

macromedia®
STUDIO 8

Retrofitting the Pet Market Application for Mach-II

An apology

By Hal Helms, Ben Edwards, Matthew Woodward

It's all Simon's fault.

We say this to all framework writers who, even now, are trying to recover from the task assigned them by CFDJ's editor-in-chief: provide an article and an implementation of the Macromedia Pet Market application in their chosen framework.

Realizing that our first sentence might not serve as sufficient explanation for those weary framework authors, allow us to provide further clarification…

It began with an innocent-seeming dinner for speakers at the recent "Fusebox and Frameworks" conference. When we found ourselves seated next to Simon Horwith, we had a great deal to talk about. During our conversation, Simon suggested that it would be very helpful to ColdFusion programmers if a reference application existed that was implemented in several different frameworks. "We could have a cfpetmarket.com," he said, "and have different framework authors write a reference application and provide an article to go along with it. For the next issue of CFDJ."

This was the point where we went wrong – badly wrong – and the reason for which an apology is due: we agreed with Simon. Given the short time frame, our indiscretion meant that we were guaranteeing that our fellow framework authors (and ourselves) were in for some late nights. For our hasty improvidence, we offer a corporate mea culpa.

Coming Up with a Plan

The three of us put our heads together. It seemed the first thing to do was to review the existing application. This presented us with a problem: frankly, we didn't like what we saw. Installation of this simple application was anything but simple. The application almost seemed to be illustrating "worst practices" including such miscues as:

- Having CFCs reference variables set in the request scope, thereby breaking encapsulation
- Poor organization of files (e.g., including detailview.cfm in a subdirectory of extensions/components/petmarket)
- Poor separation of concerns in CFCs (e.g., including login and placeOrder methods within the User.cfc.)
- Rampant use of the this scope in CFCs, disregarding information hiding
- Queries used in display pages (e.g., shoppingcart.cfm)
- Jumble of mixed scopes used in display pages
- Even the database schema was – unusual

In short, we didn't much like it. But what to do? Although it was tempting to rewrite the application from the ground up, retaining only the look and feel of the existing one, we decided that in order to focus on the ease of use of Mach-II and to help developers who are often tasked to retrofit existing applications, we would strive to keep as much of the code as possible as it was originally written.

Our decisions also led us not to integrate any of the helper frameworks such as ColdSpring or Tartan, something we surely would have done had we written the code from the ground up. In short, the code in the Mach-II version of the Pet Market should not be viewed as exemplary, but rather as the result of retrofitting an existing application – warts and all – to make use of Mach-II.

Mach-II Components

Specifically, we wanted to show how nicely Mach-II can fit into applications in its role in the Model-View-Controller (MVC) architecture. MVC is a well-accepted architectural design pattern in which the components of an application are separated by functionality.

Model components represent the business logic of the application as well as a software simulation of the inventory of the business domain (and are often called domain model components). Model components might include representations of orders, customers, products, etc.

View components form the user interface, allowing users to receive information from the application and make requests of the application. HTML pages, Flash movies, and Java applets are all examples of view components.

Mediating between model and view are controller components, receiving requests and calling on model and/or view components as needed to fulfill a request. For example, a user request to edit application preferences would entail a call to the model to determine the existing preferences and a call to the view to display a form for changing these preferences.

Many applications use some form of controller, even if nothing more than a switch statement. Frameworks such as Mach-II, Fusebox, and Model-Glue provide a more extensive controller and often include prebuilt functionality as well as "hooks" for developers who want to extend the capabilities of the framework.

Within the Mach-II world, various controller functions are assigned to different Mach-II elements. The central element in a Mach-II application is an event. An event is an all-purpose mechanism for encapsulating a request and the results (if any) of fulfilling that request. In our "edit preferences" example, Mach-II creates an event object (an instance of Mach-II's Event class) that comes with prebuilt functionality.

Among other things, events can store information related to a request. In the "edit preferences" example, Mach-II stores the results of asking the model for the user's current preferences (possibly a query) in the event. (Event objects have arguments that can be accessed with prebuilt get/setArg methods.)

When the view page – in this case a user preference form – is called, it gets any information it needs from the event (shown in the illustration as a cube). In a Mach-II application, as in any well-designed MVC application, the model and the view components exist completely separate and independent of each other. Such loose coupling, as it is called, aids in both code reuse and maintainability.

The rest of Mach-II's machinery exists to support the Event. A Listener element is a CFC that extends Mach-II's Listener class and "listens" for events, taking appropriate action for the individual event. Using our example of a user editing their preferences again, we might have a UserListener that is notified of an "editPreferences" event. It is the UserListener that actually calls the model for the user's existing preferences.

Listeners do not, though, call view pages. Doing so would decrease the listener's ability to be reused and make application maintenance more difficult. Instead, Mach-II uses a declarative programming mechanism in which the instructions for responding to a particular event are written not in code (programatically) but in some sort of simple configuration file (declaratively).

For this configuration file, Mach-II uses the file, mach-ii.xml. Different sections of the configuration file allow the developer to customize how Mach-II works for any particular application. (Rather than forcing the developer to adapt to Mach-II, Mach-II adapts to the developer.)

Mach-II's Configuration File

If you haven't done so, download the M2PetMarket application from www.mach-II.com. A subdirectory, controller, holds the mach-ii.xml file. The file has six sections:

1. ***properties:*** Used to set general application settings including such properties as applicationRoot (the directory the application files are to be found in) and defaultEvent (the event to be used when none is specified).
2. ***listeners:*** Used to register any listeners used in the application. Mach-II allows you to use a single, general purpose listener or to specify separate listeners dedicated to specific areas of interest (e.g., UserListener, ShoppingCartListener, etc.).
3. ***event-filters:*** Used to preprocess events before any listeners respond to them. Typical functionality found in event-filters would include granular security ("Can this user call this event?") and server-side form validation.
4. ***plugins:*** Used to register components that are called at various "plugin points" throughout the life cycle of an event. The Mach-II Pet Market application uses a plug-in to ensure, among other things, that a user is logged into the application.
5. ***event-handlers:*** Used to indicate how this Mach-II application should respond to a specific event. In our "edit preferences" example, the code for the editPreferences event-handler would look like this:

```
<event-handler name="editPreferences">
 <notify listener="UserListener" method="getCurrentPreferences" resultArg="
userPreferences" />
 <view-page name="userPreferenceForm" />
</event-handler>
```

6. ***page-views:*** Used to register the view pages used within this application. Pages that are registered here (providing their physical location) can then be referenced within an event-handler by (as shown in our example).

Examining the M2PetMarket

There is, of course, a DTD available for the mach-ii.xml file, but the set of elements and sub-elements is fairly simple. Let's look at the individual elements in the configuration file for the M2PetMarket application. We start with properties.

```
<properties>
    <!-- Standard Mach-II application settings. -->
    <property name="applicationRoot" value="/M2PetMarket" />
    <property name="defaultEvent" value="home" />
    <property name="eventParameter" value="event" />
    <property name="parameterPrecedence" value="form" />
    <property name="maxEvents" value="10" />
    <property name="exceptionEvent" value="exceptionEvent" />
    <!-- PetMarket application settings. -->
    <property name="dsn" value="M2PetMarket" />
    <property name="locale" value="en_US" />
</properties>
```

The applicationRoot is set to /M2PetMarket, indicating that a directory, M2PetMarket, should be a subdirectory of your Web root. The defaultEvent is home. When users initially come to the site, this is the entry point to the application. We won't change the properties, eventParameter, parameterPrecedence, maxEvents, and exceptionEvent and, for the sake of brevity, won't go into in this article.

Two additional properties, dsn and locale, are specific to this application and aren't part of the standard Mach-II properties (as the others are). Once set, these properties are available to all registered Mach-II components (listeners, plug-ins, etc.). Placing properties in the configuration file allows us to change their values, if needed, without changing any existing (and tested) code.

Next, we register listeners in their section of the configuration file:

```
<listeners>
   <listener name="StoreListener" type="M2PetMarket.m2components.StoreL-
istener">
   </listener>
   <listener name="CheckoutListener" type="M2PetMarket.m2components.
CheckoutListener">
   </listener>
</listeners>
```

Here, we've decided to use separate listeners to handle functions related to the basic store and the check out process. Remember, listeners are CFCs and different methods in those CFCs will be called depending on the event and how we wish to handle that event. We'll look at those methods when we get to the section on event handlers.

Our event-filters section is empty – this simple application requires no event filters.

We have two entries in the plugins section, PetMarketPlugin and TracePlugin. The configure method of a plugin is called automatically by Mach-II when the application is first initialized and allows us to write any initialization code. In the case of PetMarketPlugin, configure creates an instance of the Store CFC and calls its superclass, MachII.framework.Plugin, to keep this object in persistent memory.

The Store CFC has the following capabilities:

- init acts as a pseudo-constructor. It accepts a data source name (DSN) and makes it an instance variable.
- getCategory accepts a category ID and returns a properly instantiated Category object.
- getProduct accepts a product ID and returns a properly instantiated Product object.
- getProductItem accepts a product item ID and returns a properly instantiated ProductItem object.
- placeOrder accepts a User object and a ShoppingCart object and completes the placement of an order.
- queryCategories returns a recordset of all categories.
- queryProductItems returns a recordset of product items, optionally limited if an argument, productoid, is passed to this method.
- queryProducts returns a recordset of products, optionally limited if an argument, categoryoid, is passed to this method.
- searchProducts returns a recordset of products based on a keyword passed into this method.
- getDSN returns the instance variable, DSN.
- setDSN sets the instance variable, DSN.

All plugins extend the base Mach-II Plugin class, which has six plugin point methods. Individual plugins (such as PetMarketPlugin) may override any or all of these methods, depending on where the plugin developer wishes their code to called.

- preProcess is called before any event processing.
- preEvent is called before each event is handled. (A single HTTP request may encompass several events.)
- postEvent is called after each event is handled.
- preView is called before any view (i.e., display) page is rendered.
- postView is called after any view page is rendered.
- postProcess is called after all event processing.
- handleException is called when an exception occurs (before any exception event is processed).

PetMarketPlugin overrides two of these plugin point methods, preProcess and preEvent. In the preProcess method, the plugin ensures that a User object exists in the session scope. In the preEvent method, the plugin places the Store object, the User object, and the locale (a property specified in the configuration file) in the current event.

The TracePlugin is used for debugging purposes and provides information about the events processed and the time each took.

In the event-handlers section, we specify how the application should respond to various events (see Listing 1).

These events encompass the entire functionality of the PetMarket application. Let's quickly look at how each event is processed.

The home event uses a page-view element. When a page-view is specified, Mach-II looks for the appropriate display page registered in view-pages (another section in the configuration file) and displays the file(s).

The preferences, about, legal, affiliate, category, product, showCart, and search events are similarly handled, including display pages registered in the view-pages section of the configuration file.

The updateUserPreferences event is a bit more interesting. It notifies a listener, StoreListener, that an event has occurred and

Is the Store CFC a listener? We did register a StoreListener, but pointed it at another CFC. Readers may understandably be confused as to why some CFCs are listeners while others, like Store.cfc, are not.

Unfortunately, we don't have space for a full discussion of the differences, but the reasons lie in the nature of MVC applications. Model components, we said, represent the business domain. The elements of the Model are, to some degree, application-agnostic. A Store represents a basic part of the inventory of Pet Market, the business. The Pet Market application may be only one of many applications and some of these other applications may need access to the Store. Listeners, on the other hand, are very specific to a particular application and any reusability is strictly a bonus.

passes the Event object to that listener's updateUserPreferences method. Before this, though, it specifies an event-mapping. This requires some explanation.

To understand what event-mappings are and how they operate, look at the updateUserPreferences method of the StoreListener shown in Listing 2.

This method receives an argument of type, Event. In fact, virtually all listener methods receive this argument. The method then sets the User object's email, favoritepet, mailings_sales, and mailing_tips instance variables. When this is done, the listener places a status message in the event and is done with its work.

Where do we go from here? The original Pet Market application returns the user to the preferences page. Given this, we could announce a new event, preferences, from within the listener, since Listener objects can announce events. But to do so would tightly couple this listener with the flow of the application – something that would violate the very sound principle of loose coupling. Instead, the Listener object announces something quite generic, userPreferencesUpdated.

However, we have no userPreferencesUpdated event registered in our configuration file, nor do we wish to. Instead, we want to substitute our previously registered preferences event for the announced userPreferencesUpdated event, and that's just what using event-mapping allows us to do. When the StoreListener announces userPreferencesUpdated, Mach-II intercepts the event announcement, substituting in its place the event, preferences.

The event handler for addItemsToCart notifies the StoreListener object, calling its addItemsToCart method. The addItemsToCart method within the listener announces a cartUpdated event, for which we have a corresponding registered event.

The event handler for updateCart notifies the StoreListener object, calling its updateCart method. Again, the listener announces an event, cartUpdated. We have a registered cartUpdated event that introduces a new Mach-II XML configuration element, redirect. The redirect tag acts like the ColdFusion tag, cflocation, causing the browser to make a new HTTP request. The redirect tag is used to ensure that if the user clicks the "refresh" button on their browser, the updateCart method will not be called again.

Checkout is handled by the original application by checking the value of a variable, step, that indicates the point in the multi-step checkout process the user is currently in. If we had designed this application originally, we would have used discrete events for each step. Then, if the flow of the checkout were to change later, the change would have been much less disruptive. But we chose to keep the original mechanism intact and so have a startCheckout method.

The event, startCheckout, uses the Mach-II XML configuration element, event-arg, to set a variable,step, in the current event's eventArgs structure. It then displays the checkout display page.

The event, continueCheckout, notifies the CheckoutListener, calling its continueCheckout method. The continueCheckout method has quite a bit of conditional code that, depending on the value of step, performs different functions.

The event, checkoutStep, is a private event, meaning that it cannot be called as part of a new HTTP request – as part of a URL's query string, for example. Events that should not be user-callable (loginValidated, for example) should be marked private.

The checkoutStep event notifies the CheckoutListener, calling its getStatesList. Notice the use of the resultArg attribute, which places the results returned by CheckoutListener's getStatesList method into the current event.

This same event causes CheckoutListener to be notified again, this time calling its getShippingMethods and placing the results of that method call into the current event. Finally, a display page, registered as checkout, is rendered.

When the user wishes to retrace their steps in the checkout process, the previousCheckout event is called. This announces the private event, checkoutStep.

When checkout is completed, the event, completeCheckout, notifies the CheckoutListener, calling its completeCheckout method. (Note that the cases shown where the event name and the listener's method name are the same is not a requirement; it simply made sense to use the same name.)

Within the CheckoutListener's completeCheckout method, another method, checkoutComplete is announced. Again, we use the Mach-II redirect XML element so that a "refresh" button press will not cause the completeCheckout to be called again.

The checkoutReceipt event sets the step variable within the current event to 7 and displays the multi-purpose checkout page.

Our final event, exceptionEvent, will be used whenever the Mach-II framework encounters an untrapped error. Mach-II creates a new event (with the cfcatch information inside of it) and automatically announces exceptionEvent. Application developers will usually want to change how exceptionEvents process exceptions, as the default event handler just displays exception information.

Finally, we arrive at our final section, page-views. In this section, individual display pages are registered. Views are registered so that a change to the path of a view page only needs to be updated once rather than each time the page is used. Page views are given a name (used in the event-handler's view-page element) and a page (the physical location of the file).

```
<page-views>
    <page-view name="index" page="/views/index.cfm" />
    <page-view name="preferences" page="/views/preferences.cfm" />
    <page-view name="about" page="/views/about.cfm" />
    <page-view name="legal" page="/views/legal.cfm" />
    <page-view name="affiliate" page="/views/affiliate.cfm" />
    <page-view name="category" page="/views/category.cfm" />
    <page-view name="searchResults" page="/views/searchResults.cfm" />
    <page-view name="cart" page="/views/cart.cfm" />
    <page-view name="checkout" page="/views/checkout.cfm" />
    <page-view name="exception" page="/views/exception.cfm" />
</page-views>
```

Conclusion

Recently, one of us received an e-mail that stated, "I'd really like to use Mach-II, but it seems like it's meant strictly for OO gurus and that's not me, at least not yet." While we hope we've given you a sense of how to retrofit an existing application, we really hope we've given you a sense that Mach-II is not just for

"OO gurus" or gurus of any kind, for that matter.

Mach-II was created to help working programmers with the problems of maintaining applications as they evolve to increasing complexity. As this issue shows, the growing sophistication of the ColdFusion community has made a number of frameworks available. That's a very encouraging sign and – all kidding aside – we salute Simon for bringing together all framework writers to "lay out their case" for each developer to judge for themselves.

And so, ladies and gentlemen of the only jury that matters, we present (in no particular order) our "Top 15 Benefits of Using Mach-II".

1. *No more spaghetti code*. This makes your applications much easier to write, debug, and maintain. Mach-II by design strongly encourages developers to do things "the right way."
2. *Elimination of scope issues*. Because all elements in a Mach-II application are geared to get their information from the event, there is no need to worry about whether a particular variable came from the form, URL, request, variables, etc., scope. This makes writing applications clear and simple since all the variables are always in the same place.
3. *Team developmen*t. The use of any standardized framework helps teams of developers work together to build software. Mach-II, with its emphasis on separation of concerns among elements, a declarative programming framework, and the use of public APIs, goes the next step in empowering teams.
4. *End to page-centric development*. Mach-II's use of the design pattern known as "Front Controller" relieves developers of the fragility of page-to-page application flow. Announce the event and Mach-II handles the rest.
5. *Code organization*. Mach-II encourages excellent code organization but doesn't force developers to adopt any particular conventions or standards. A Mach-II application skeleton (available from www.mach-ii.com) shows clear model and view directories, but other organizational schemes can easily be used.
6. *Low-impact maintenance*. Mach-II is designed to use the tried-and-true MVC design pattern. MVC separates presentation from logic in the code, thereby making the application flexible and very easy to maintain since changes in one area are localized without breaking the rest of your application.
7. *Encourages object-oriented programming*. We think that a mastery of OOP is an absolute necessity for career survival. With the OO tools available in ColdFusion, Mach-II is a perfect way to begin mastering object-orientation in a language you're already familiar with.
8. *Ease of installation*. Mach-II is just a CF application. Just drop a single copy of the Mach-II code in a Mach-II directory under your Web root and you're ready to go.
9. *Built on proven principles*. Mach-II is based on solid software engineering principles (event-driven, implicit invocation) that have long proved effective in software development. Mach-II brings this power, flexibility, and simplicity to ColdFusion developers.
10. *Code reuse*. Mach-II's separation of concerns allows the same model components to be reused across many applications within an organization.
11. *Portal development*. Powerful view capabilities allow for simple development of templates or portal-style applications. Multiple views can be built up within a single event and presented to the user at the end of the event.
12. *Framework extensibility*. Plugins and filters are extremely powerful, flexible, easy-to-use application components that can handle application concerns such as logging, debugging, security, etc., at the event level without concerning listeners or, worse, business objects. If you need system-wide functionality, you can extend the framework without touching the core files.
13. *Mature and stable*. Mach-II is the most mature, stable OO framework for ColdFusion, and has been proven in both small and very large applications in organizations as diverse as the Federal Reserve Board, Dow Jones, and ColdFusion's parent, Macromedia.
14. *Separation of concerns*. Underpinning many of the benefits listed above, this fundamental software engineering principle is the foundation on which Mach-II is built. Page view elements are unaware of listener and model elements. Model components know nothing of either listeners or page views – or even of Mach-II. By adherence to the twin principles of loose coupling and tight cohesion, Mach-II applications are built to withstand the changes over time inherent in successful software deployments.
15. *Integration with other frameworks*. If you've looked at frameworks like ColdSpring and Tartan, you've probably been impressed with them. We are too. Mach-II's plugin architecture makes it a perfect partner with both current frameworks and others that may appear.

If you're ready to give Mach-II a spin, come to www.mach-ii.com. There you'll find documentation, sample applications, even video tutorials for building Mach-II applications.

About the Authors

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.

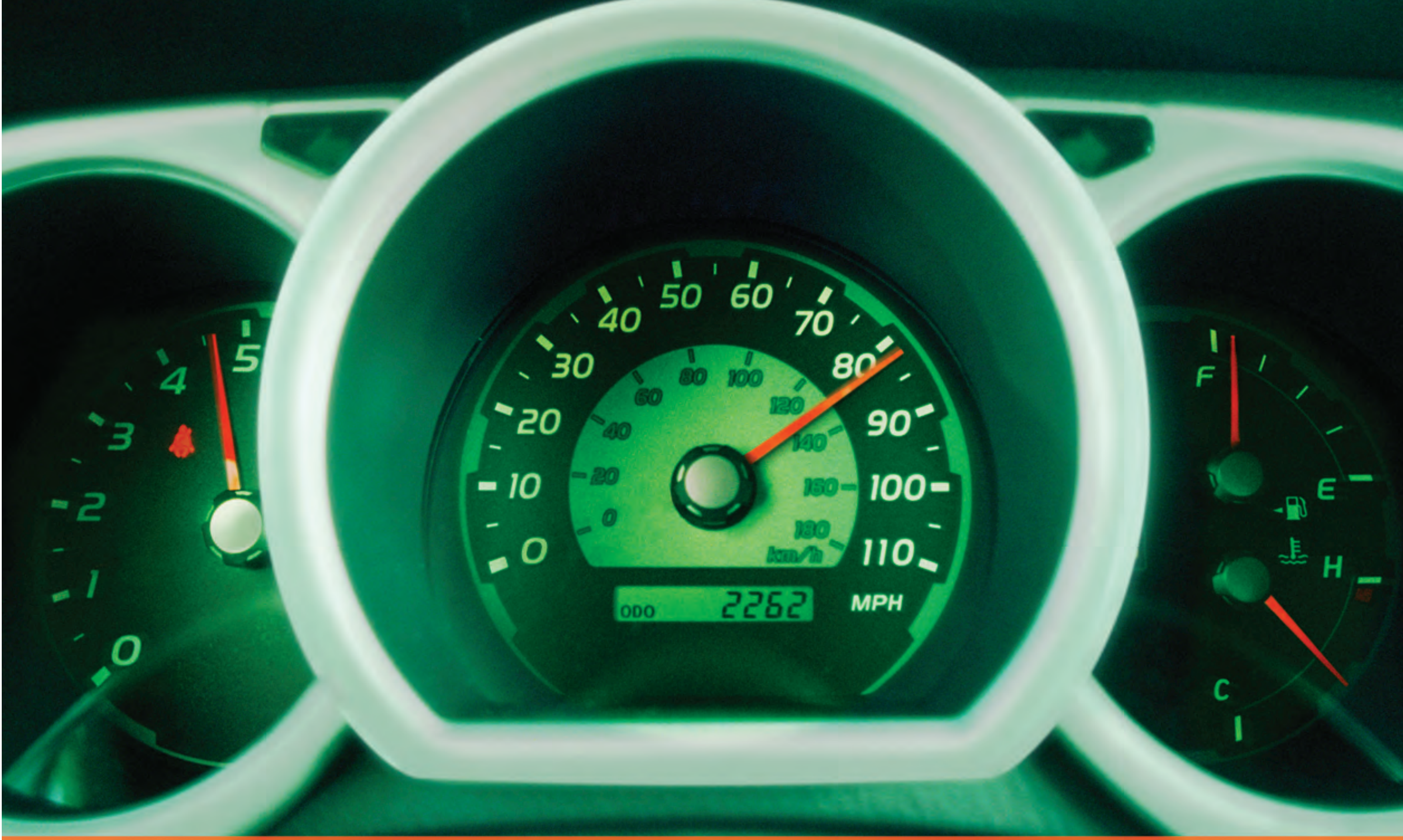
Ben Edwards is a Sun Certified Java Programmer and holds a degree in computer science from the Georgia Institute of Technology. He currently trains developers on software engineering practices focusing on Java, object-oriented programming, and software architectures. Ben is also cofounder of the Mach-II project.

Matt Woodward is a Web application developer for i2 Technologies in Dallas, Texas, and also works as a consultant through his company, Sixth Floor Software. He is a Macromedia Certified ColdFusion Developer, a member of Team Macromedia, and has been using ColdFusion since 1996. In addition to his ColdFusion work, Matt also develops in Java and PHP

hal@halhelms.com

ben@ben-edwards.com

mpwoodward@mac.com



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

macromedia® FLASH VIDEO STREAMING SERVICE

Call (800) 254-7554
Visit www.vitalstream.com

VitalStream®

Listing 1

```
<event-handlers>
    <event-handler event="home" access="public">
        <view-page name="index" />
    </event-handler>

    <event-handler event="preferences" access="public">
        <view-page name="preferences" />
    </event-handler>

    <event-handler event="about" access="public">
        <view-page name="about" />
    </event-handler>

    <event-handler event="legal" access="public">
        <view-page name="legal" />
    </event-handler>

    <event-handler event="affiliate" access="public">
        <view-page name="affiliate" />
    </event-handler>

    <event-handler event="category" access="public">
        <view-page name="category" />
    </event-handler>

    <event-handler event="product" access="public">
        <view-page name="category" />
    </event-handler>

    <event-handler event="search" access="public">
        <!-- Right now the search functionality is performed in page
(via a call to store.search()), but it should probably be moved to a
listener call.-->
        <view-page name="searchResults" />
    </event-handler>

    <event-handler event="updateUserPreferences" access="public">
        <!-- We'll notify the listener to update the user preferences.
When they're updated, 'userPreferencesUpdated' event is announced.
When the event is announced, we want to map it to 'show preferences'
event. -->
        <event-mapping event="userPreferencesUpdated"
mapping="preferences" />
        <notify listener="StoreListener" method="updateUserPreferences
" />
    </event-handler>

    <event-handler event="showCart" access="public">
        <view-page name="cart" />
    </event-handler>

    <event-handler event="addItemsToCart" access="public">
        <!-- Notify the listener to add the item to the cart.
When updated, 'cartUpdated' event is announced. -->
        <notify listener="StoreListener" method="addItemsToCart" />
    </event-handler>

    <event-handler event="updateCart" access="public">
        <!-- Notify the listener to update the cart. When updated,
'cartUpdated' event is announced. -->
        <notify listener="StoreListener" method="updateCart" />
    </event-handler>

    <event-handler event="cartUpdated" access="private">
        <redirect event="showCart" args="statusMessage" />
    </event-handler>

    <event-handler event="startCheckout" access="public">
        <event-arg name="step" value="1" />
        <view-page name="checkout" />
    </event-handler>

    <event-handler event="continueCheckout" access="public">
        <!-- Notify the listener to persist the current step's data.
When updated, 'checkoutStep' event is announced (with the step incre-
mented). -->
        <notify listener="CheckoutListener" method="continueCheckout"
/>
    </event-handler>

    <event-handler event="checkoutStep" access="private">
        <notify listener="CheckoutListener" method="getStatesList"
resultArg="qryStateList" />
        <notify listener="CheckoutListener" method="getShippingMethods
" resultArg="qryShippingMethods" />
        <view-page name="checkout" />
    </event-handler>

    <event-handler event="previousCheckout" access="public">
        <announce event="checkoutStep" />
    </event-handler>

    <event-handler event="completeCheckout" access="public">
        <notify listener="CheckoutListener" method="completeCheckout"
/>
    </event-handler>

    <event-handler event="checkoutComplete" access="private">
        <!-- Redirect to the 'checkoutReceipt' event so hitting reload
won't re-order. -->
        <redirect event="checkoutReceipt" />
    </event-handler>

    <event-handler event="checkoutReceipt" access="public">
        <event-arg name="step" value="7" />
        <view-page name="checkout" />
    </event-handler>

    <event-handler event="exceptionEvent" access="private">
        <view-page name="exception" />
    </event-handler>
</event-handlers>
```

Listing 2

```
<cffunction name="updateUserPreferences" access="public"
returntype="void" output="false" hint="">
    <cfargument name="event" type="MachII.framework.Event"
required="true" />
    <cfset var user = event.getArg('user') />

    <!--- Update the user's preferences. --->
    <cfset user.setEmail( event.getArg('email') ) />
    <cfset user.setPreference('favoritepet', event.
getArg('favoritepet')) />
    <cfset user.setPreference('mailings_sales', event.
getArg('mailings_sales','0')) />
    <cfset user.setPreference('mailings_tips', event.getArg('mailings_
tips','0')) />

    <!--- Set a status message in the event. --->
    <cfset arguments.event.setArg('statusMessage', 'Your preferences
have been saved.') />

    <!--- Announce an event to let the app know the user's preferences
have been updated. --->
    <cfset announceEvent('userPreferencesUpdated', arguments.event.
getArgs()) />
</cffunction>
```

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

HOSTING.COM

THE FIRST NAME IN HOSTING

Developers!

Developers!

Developers!

We Love Them!

SERVE. SUPPORT. SECURE. STORE.

www.hosting.com

The ColdSpring Framework

Pet Market implementation

By David Ross & Chris Scott

ColdSpring is a framework for ColdFusion Components, inspired by the Spring Framework for Java, and its core focus is to manage the dependencies within your CFC "model."

Dependencies are more common than you think: when one CFC needs another CFC to perform a task, it depends on that other CFC (and those two CFCs are known as "collaborators"). When a CFC needs a piece of configuration data, such as a datasource name, it depends on that piece of data. ColdSpring enables you to declaratively supply your CFCs with their dependencies, freeing you from having to write the code to do so (which unfortunately ends up in just about each and every component). ColdSpring practices what many refer to as "inversion-of-control." This simply means that the control of creating objects and resolving their dependencies is lifted out of the code and into the code's "container," which in this case is ColdSpring. "Dependency-Injection" is probably a better term to describe what ColdSpring does: it injects dependencies into your CFCs as they are created.

Also part of ColdSpring is the first aspect-oriented-programming (AOP) framework for CFCs (which we'll cover in further detail shortly).

For building an application like the Pet Market, ColdSpring would typically be used in conjunction with one of the other MVC application frameworks like Fusebox, Mach-II, or Model-Glue. However, for the purposes of this article, we were unable to use one of those frameworks, so we took the approach of attempting to retrofit a brand new ColdSpring-managed model on top of the existing Pet Market application. This means that our version of the Pet Market should be used to examine how ColdSpring manages a model regardless of the application framework used. (The code for this article can be downloaded from cfpetmarket.com.)

Figure 1 provides a picture of where ColdSpring sits in the overall Pet Market application architecture.

Figure 1 is an example of typical MVC (Model-View-Controller) architecture. The "Service," "DAO," "Gateway," and "Utility" components are part of our Pet Market model. Each component is "managed" by ColdSpring, meaning ColdSpring is responsible for both creating the component and resolving its dependencies. When the controller layer needs to talk to a model component, it simply asks ColdSpring for the component. After retrieving a component from ColdSpring, the controller layer would call methods directly on that component, so ColdSpring doesn't really "sit between" the controller and the model.

To use ColdSpring with your application, you need to write some code to create a ColdSpring "BeanFactory". The BeanFactory is typically the only ColdSpring CFC that you will interact with, thus setting up ColdSpring for use within an application is trivial. It's worth mentioning that none of this is necessary if you are using the Mach-II or ModelGlue application frameworks, as both offer hooks to handle creating the ColdSpring BeanFactory automatically. However, to see how we set up ColdSpring in the Pet Market application, take a look at our application.cfm, the relevant part of which is shown here:

```
<cfif not structKeyExists(application,"beanFactory")
         or structKeyExists(url,"reloadApp")>
   <cflock name="PetMarket_Startup" timeout="25">
         <cfif not structKeyExists(application,"beanFactory")
                  or structKeyExists(url,"reloadApp")>
                  <cfset application.beanFactory =
createObject('component',
                    'coldspring.beans.DefaultXmlBeanFactory').
init()/>
                  <cfset application.beanFactory.
loadBeans(expandPath("./components.xml"))/>
         </cfif>
   </cflock>
</cfif>
```

All we are doing here is creating the ColdSpring BeanFactory and placing it in the application scope. The <cfif/> and <cflock/> blocks are necessary to ensure that the BeanFactory starts up in a thread-safe manner, because as you can see from the code, we are storing the BeanFactory in the application scope and we want to make sure that all requests wait while the BeanFactory is created (this only happens once during application "startup"). In order to tell ColdSpring about your CFCs, we need to supply it with a set of "bean definitions," which are written in a simple XML format. We named our configuration file components.xml. You can see this done in the code above, where we are passing components.xml's file path into the BeanFactory using the loadBeans() method. Here's a quick sample of our bean definitions:

WE HELP YOU DELIVER THE WHOLE PACKAGE

VERIO SUPPLIES THE BEST TECHNOLOGY, RESOURCES, AND EXPERTISE TO SUPPORT YOUR .NET SOLUTIONS

When you partner with Verio, you can deliver complete solutions for your customers' business needs. As your trusted expert advisor for managed hosting, we support your application development with our managed services and infrastructure from state-of-the-art Verio data centers.

As a viaVerio® partner, you'll enhance your solutions portfolio and add value to your business by gaining access to our suite of global products and services, including managed storage and security. You'll also gain peace of mind in knowing that Verio is backed by the financial and operational excellence of NTT Communications, the world's largest telecommunications company.

BUILD YOUR BUSINESS BY PARTNERING WITH THE MANAGED HOSTING INFRASTRUCTURE EXPERTS. Nothing turns on the power of the Internet like Verio products and services.

CALL 866.237.4121 today to find out more about how a viaVerio partnership can complete your business offering. Or visit: www.verio.com/partners

Power. Performance. Results.

VERIO
An NTT Communications Company

viaVERIO℠ Business Partner

```
<beans>
   <!-- Session Service Definition -->
   <bean id="SessionService"
         class="petmarket.util.UsageSessionService" />

   ...

   <!-- User Service -->
   <bean id="UserService"
         class="petmarket.component.user.UserService">
    <!-- needs a UserDAO -->
    <property name="UserDAO">
           <bean id="UserDAO"
                           class="petmarket.component.user.UserDAO"/>
    </property>
    <!-- also needs the session service -->
    <property name="SessionService">
           <ref bean="SessionService"/>
    </property>
   </bean>
</beans>
```

Use <bean/> tags to define your CFCs. You may have heard the term “bean” used in a number of different contexts related to CFC development, but here it’s chosen because ColdSpring expects that, in order to take advantage of certain

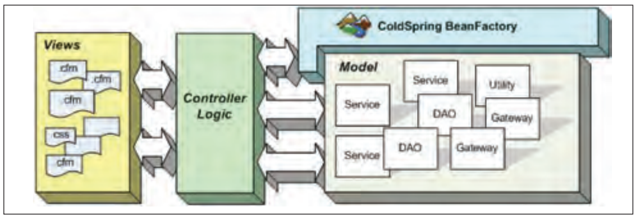


Figure 1

features, you adhere to the Java programming language’s JavaBean specification/conventions. Now we will take a look at the XML above in detail. First we defined a <bean/> and gave it an ID of “SessionService”. The class attribute, “petmarket.util.UsageSessionService”, tells ColdSpring which CFC it should use when asked for the “SessionService”. You will also see the “UserService” <bean/> definition, but this one is a bit more complex. We define the ID and class just like our SessionService, however, we also define some <property/> tags within the <bean/> tag. First we define a “UserDAO” <property/> (DAO stands for “Data Access Object”, it’s part of the gritty details of our Pet Market model but has nothing to do with ColdSpring itself). Whatever you put between the <property></property> tags will be injected, by ColdSpring, when the CFC is actually created. Thus, we have defined an “inner-bean” within the UserDAO property, meaning ColdSpring will create a UserDAO CFC and pass that into the UserService when it is created. You must provide ColdSpring with a way to do this, and, in the case of <property/>, ColdSpring expects there to be a “setter method” that will accept the UserDAO, and it must be named “setUserDAO”. This is where ColdSpring relies on the JavaBeans spec, meaning that you adhere to the JavaBean convention of providing setter methods for public properties. The setter method within our UserService that accepts the UserDAO looks like this:

```
<cffunction name="setUserDAO" access="public" returntype="void"
output="false"
     hint="Dependency: userDAO">
   <cfargument name="UserDAO" type="petmarket.component.user.UserDAO"
required="true"/>
   <cfset variables.userDAO = arguments.UserDAO />
</cffunction>
```

There is nothing special going on here: the UserService just takes the supplied UserDAO and retains a reference to it (by placing it in its variable’s scope). You don’t have to use <property/> tags and setter methods to expose your CFCs to dependency injection: ColdSpring also supports a <constructor-arg> tag. The <contructor-arg/> tag tells ColdSpring to inject dependencies as arguments to a CFC’s init() method, which is also known as a “constructor.” You have to provide the <cfargument/> tags so that ColdSpring can pass in your CFC’s dependencies, but otherwise it works the same as <property/> and setter methods.

The second <property/> tag within our UserService definition is a bit different. Our UserService implementation needs a SessionService in order to perform certain tasks, but so do other components within our model. We don’t define the SessionService within the UserService as we did with the UserDAO, because the UserService isn’t the only CFC that needs to use the SessionService. Thus, we use the <ref/> tag to supply the UserService with the session service. The <ref/> tag basically just instructs ColdSpring to use a component that has been defined elsewhere within the BeanFactory (the order of the definitions doesn’t matter, e.g., you don’t have to define the SessionService before the UserService in order to use the SessionService in a <ref/> tag).

We used the same techniques to define the rest of our Pet Market model within ColdSpring (look at components.xml to see it in its entirety). All of the application’s components are wired together using ColdSpring, and there are some pretty complex dependencies that would have been very difficult to implement without ColdSpring’s help. For instance, our “ProductService” depended on several components in order to function correctly, one of them being a “ProductItemDAO”. Now, what the ProductItemDAO is or does is irrelevant to ColdSpring; it’s just part of our model. What is relevant is that the ProductItemDAO depended on the ProductService, just as the ProductService depended on the ProductItemDAO. They both needed to be able to call methods on each other, and this situation is known as a “circular dependency.” Here’s a snippet of the XML that defines this complex situation:

```
<bean id="ProductService" class="petmarket.component.product.ProductSer-
vice">
```

```
    ...
    <property name="ProductItemDAO">
     <ref bean="ProductItemDAO"/>
    </property>
</bean>

<bean id="ProductItemDAO" class="petmarket.component.product.ProductItem-
DAO">
    <property name="ProductService">
     <ref bean="ProductService"/>
    </property>
</bean>
```

ColdSpring actually supports circular dependency resolution when you are using setter-based injection (e.g., you are using <property/> instead of <constructor-arg/>), so the above is perfectly legal.

After the ColdSpring BeanFactory is loaded with BeanDefinitions, your controller layer code would ask for a ColdSpring "Bean" (remember, it's just a CFC) using the BeanFactory's getBean() method. You pass the ID of the Bean if you want to getBean(), meaning that if you wanted to get the SessionService defined above, you would call getBean('SessionService'). You can actually see this taking place in our Pet Market's application.cfm, after we created and configured the BeanFactory:

<cfset request.SessionService = application.beanFactory.getBean("SessionService") />

Within our Pet Market model, when we got to the OrderService, we realized that some of our business requirements were pretty complex. To save an order we need to perform multiple tasks, save the gathered user data, save the purchased items, and update the inventory. We also decided it may be a good idea to store a backup of the order in a log file, but we were not sure if this is a concrete business requirement. This is where using ColdSpring in a "service-oriented" architecture really shines. If you take a look at the OrderService, you see that it contains setters for the UserService and ProductService, as well as the OrderDao it uses to save the actual order items to the database. When placeOrder(user,cart) is called on the OrderService, saveUserForOrder(user) is called on the userService, placeOrder(cart) is called on the OrderDao, and updateProductQuantities(cartItems) is called on the ProductService. The actual transaction management is not handled in any of the DAOs, as this would make it impossible to propagate such a complex business process. Instead the transaction is managed by the OrderService component. In a larger application, you may have many complex business

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.

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.

processes, and without using a container like ColdSpring to manage collaborators, you may end up writing components that perform multiple tasks at once, which will unfortunately make them far less reusable and coherent.

The one thing you may have noticed that was left out of this process was logging a record of the order. This brings up a pretty tricky issue, because logging the order is not necessarily part of the business logic; it's not really even a business requirement. What we would like to do is add or remove this feature declaratively, instead of adding any code to our existing services. Luckily ColdSpring's new aspect-oriented programming framework allows us to do exactly this. Before we get into exactly how this is implemented, a quick overview of aspect-oriented programming is in order. Since a full discussion of aspect-oriented programming is outside the scope of this article, we'll briefly describe the steps involved, and include links to resources at the end of the article.

Aspect-oriented programming assists in applying certain types of "cross-cutting" functionality broadly across model components. Functionality like logging is not necessarily part of the business logic of a UserService component, but we may still want to add logging to certain methods within that component. AOP also allows us to write logging code in one place and through configuration apply it to one or many of our model components. In order to use AOP to log an order, we will write an Advice component that contains the logging code, configure an instance of a ColdSpring "Pointcut Advisor" component that will contain the Advice and identify the names of methods to apply it to, and configure a ColdSpring ProxyFactoryBean component to create a new instance of a supplied target object with the now advised methods. The process of creating a new component that combines the code in the Aspects with the code in the target object is known as weaving. There are a few different methods of weaving in AOP, but we use a system of dynamic runtime compilation to create a proxy object that contains the target object and the Aspects. This proxy object will be the same type as the target object, so from the perspective of the rest of your application, it will appear to be the original. The proxy object will also contain all of the same methods as the original, but each method call will check for a match from the Pointcut, and run through any Advice configured before, after or around the method call if necessary. So let's take a look at our components.xml file to see how this works in practice. First we have the definition of the OrderServiceTarget, which is the OrderService component.

```
<bean id="OrderServiceTarget"
class="petmarket.component.order.OrderService">
<property name="TaxRate"><value>.08</value></property>
<property name="ShippingMethodGateway">
<bean id="ShippingMethodGateway"
class="petmarket.component.order.ShippingMethodGateway"/>
</property>
        <property name="OrderDAO">
            <bean id="OrderDAO"
                class="petmarket.component.order.OrderDAO">
            </bean>
</property>
<property name="ProductService">
            <ref bean="ProductService"/>
        </property>
   <property name="UserService">
            <ref bean="UserService"/>
</property>
</bean>
```

Here we see the service configured with a value for the tax rate, inner beans for the ShippingMethodGateway and OrderDAO components, and references to the Product and User Services, as we discussed before. Next, we'll look at the configuration of the logOrderAdvisor, which is an instance of a ColdSpring "NamedMethodPointcutAdvisor":

```
<bean id="logOrderAdvisor" class="coldspring.aop.support.NamedMethod-
PointcutAdvisor">
<property name="advice">
   <bean id="logOrderAdvice" class="petmarket.aspects.LogOrderAdvice">
        <property name="filename">
                <value>petstoreOrders</value>
        </property>
   </bean>
   </property>
   <property name="mappedNames">
    <value>placeOrder</value>
   </property>
</bean>
```

The logOrderAdvisor has two properties: the Advice to configure and the mappedNames, which are the method names it will try to match. In this application, the advice is written specifically for the placeOrder method and that is what we supply as the mappedNames. However, more generalized logging advice could be applied just as easily to all methods by supplying "*" for mappedNames. The Advice components you write must extend one of the provided coldspring.aop.Advice components: BeforeAdvice, AfterReturningAdvice, ThrowsAdvice, and MethodInterceptor, which is also known as "around advice." For this application we've used an AfterReturningAdvice, so that we can check the value returned from placeOrder() and only log successful orders. The last piece of configuration is for the ProxyFactoryBean to generate the proxy object.

```
<bean id="OrderService" class="coldspring.aop.framework.ProxyFactory-
Bean">
   <property name="target">
    <ref bean="OrderServiceTarget" />
   </property>
   <property name="interceptorNames">
    <list>
        <value>logOrderAdvisor</value>
    </list>
   </property>
</bean>
```

“ColdSpring is ‘non-invasive,’ meaning your model components will have no idea they are being managed by ColdSpring”

Here we simply supply the target object and a list of any configured Advisors you would like to use on the component. It is worth noting that if you skip the configuration of a NamedPointcutAdvisor and simply put the name of a configured Advice bean in the list of interceptorNames, the ProxyFactory will automatically create a DefaultPointcutAdvisor for you, meaning that you would like to apply this advice to all methods in the target obejct. We give the ProxyFactoryBean the ID of OrderService, and when your application requests that service component with getBean(‘OrderService’), all the details of creating the proxy and wiring up all the necessary components will be handled for you, and, of course, as noted earlier, your application will treat the proxy as if it is the target component. It’s also worth noting that the life cycle of these components are also completely managed for you, so successive calls to getBean() will not generally create new instances of these components, unless you have configured the bean to do so, as a non-singleton.

Hopefully this article and the accompanying code will give you a good idea of what ColdSpring intends to do: it manages your model. ColdSpring is “non-invasive”, meaning your model components will have no idea they are being managed by ColdSpring. ColdSpring’s features enable you to write components that are easier to reuse and test, because they don’t contain “hard-coded” knowledge of how to resolve their own dependencies. ColdSpring enables you to declaratively configure an application, and application maintenance becomes easier because you can quickly swap out your components or transparently add behavior via AOP. We believe that as your codebase gets larger, ColdSpring will be the tool you use to keep your model under control.

About the Authors

David Ross is a ColdFusion Developer

Chris Scott is a ColdFusion Developer

dave.ross@gmail.com

asstrochris@comcast.net

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

Pet MarketFB

Petmarket a la Fusebox

By Jeff Peters

Simon says, "Build the Pet Market app in Fusebox." Simon says, "Write an article about it."

So I'm sitting on a plane at 35,000 feet, somewhere over the heartlands, writing an article. Having been the driving force behind the creation of the Wegot Widgets reference application project for Fusebox, I think the idea of an "All Pet Market" issue of *CFDJ* is a great idea.

I'm sure each of the application authors are taking a slightly different approach, even with the various frameworks taken out of the picture. Here are the ground rules by which I played when building Pet MarketFB:

1. No changes to the database
2. Use the existing application's HTML
3. Follow the whole Fusebox Lifecycle Process (FLiP)

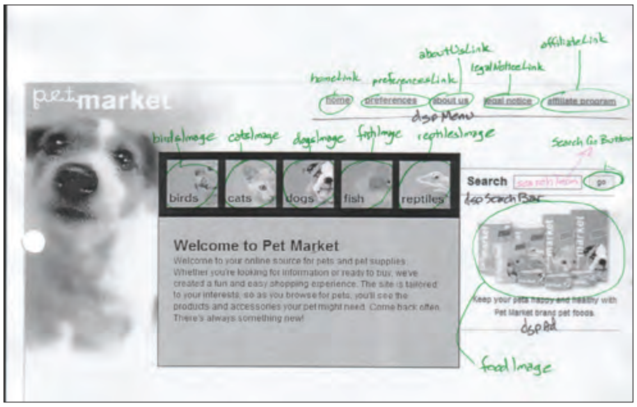
Consequently, there are a lot of things you won't see in the Pet MarketFB code. You won't see any attempt to modernize the HTML, use CSS, build accessibility, etc. While these are all laudable goals, they are beyond the scope of this project and article. In my opinion, the point of the project is to show you how the same application's code would look if arranged in Fusebox.

Prototype/Front End

Fortunately, FLiP is well-suited for an effort like this. We can pick up the project at the end of the Prototype/Frontend phase--the point called "Prototype Freeze". We just use the HTML generated by the existing application as the frozen frontend. Based on that assumption, I started by printing out the screen shots of the application and marking up the printouts as I would for any Fusebox application. The printouts are marked in various colors representing dynamic data, exit points, exit data points, and fuse file names (for the display fuses implied by each printed page). An example of a marked up page is shown in Figure 1.

Mind Mapping

Once the markup was completed, I fired up Mind Mapper Pro and started building out the architecture of the application. For each page in the markup, I chose a circuit to contain the logic represented by the markup. Each circuit is assigned fuseactions implied by the pages and the actions they require, and each fuseaction is in turn assigned one or more



Welcome to Pet Market

Welcome to your online source for pets and pet supplies. Whether you're looking for information or ready to buy, we've created a fun and easy shopping experience. The site is tailored to your interests, so as you browse for pets, you'll see the products and accessories your pet might need. Come back often. There's always something new!

Keep your pets happy and healthy with Pet Market brand pet foods.

Figure 1: Marked page photo

fuses. I also defined Exit Fuseactions and Component Content Variables. At this stage of the game, we're not concerned about code at all. It's all about the application architecture. A shot of one circuit in the finished mind map is shown in Figure 2. The complete mind map is available in the companion zip file for this article, which can be downloaded from cfpetmarket.com.

Fusedocs

With the mind map's nodes all built out, I turned my attention to the Fusedocs. Fusedocs define the responsibilities, properties, and IO for each fuse in a standardized XML vocabulary. Still, we're not working in code. The project might be completed in ColdFusion, but it could also be coded in PHP, Lasso, or any other language for which the Fusebox 4 core files have been created. An example of one of the Fusedocs is shown in Figure 3.

The purpose in a Fusedoc is to provide everything the coder needs to write the fuse, and nothing more. The responsibilities section describes the fuse's job, and the properties and io sections provide specific data about the fuse, its inputs, and its outputs.

Fusedocs to Fuse Stubs

In addition to the XML-based Fusedocs, I also added some extra stuff for some of the fuses. In particular, I copied relevant chunks of HTML from the front end into each display fuse,

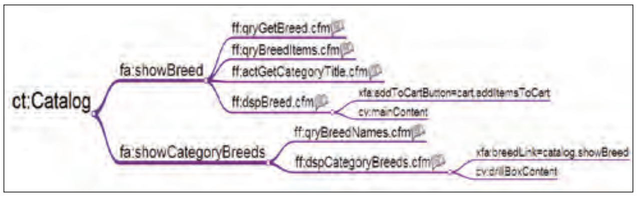
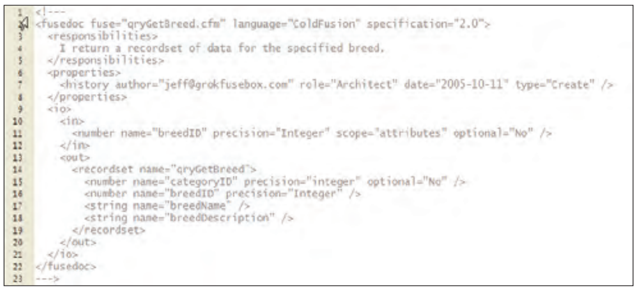
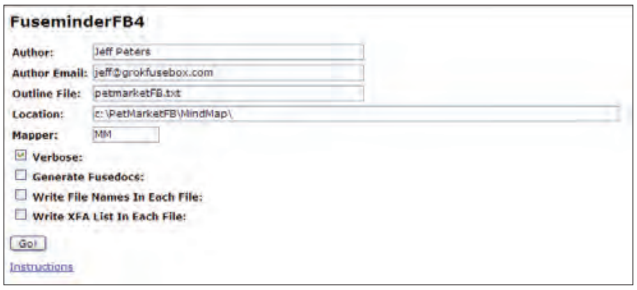


Figure 2: Mind Map



```
<!---
<fusedoc fuse="qryGetBreed.cfm" language="ColdFusion" specification="2.0">
  <responsibilities>
    I return a recordset of data for the specified breed.
  </responsibilities>
  <properties>
    <history author="jeff@grokfusebox.com" role="Architect" date="2005-10-11" type="Create" />
  </properties>
  <io>
    <in>
      <number name="breedID" precision="Integer" scope="attributes" optional="No" />
    </in>
    <out>
      <recordset name="qryGetBreed">
        <number name="categoryID" precision="integer" optional="No" />
        <number name="breedID" precision="Integer" />
        <string name="breedName" />
        <string name="breedDescription" />
      </recordset>
    </out>
  </io>
</fusedoc>
--->
```

Figure 3: Sample Fusedoc



FuseminderFB4

Author:	Jeff Peters
Author Email:	jeff@grokfusebox.com
Outline File:	petmarketFB.txt
Location:	c:\PetMarketFB\MindMap\
Mapper:	MM

✓ Verbose:
☐ Generate Fusedocs:
☐ Write File Names In Each File:
☐ Write XFA List In Each File:

Figure 4: FuseminderFB4 form

and I created a QuerySim for each query fuse. QuerySims are an easy way to provide recordsets with test data before the database is available. During the architecture phase, I'm not concerned about the database, even if one already exists – I'm concerned about satisfying the data requirements represented by the front end. The front end is the customer's expectation and it must be met. I'll worry about the database's demands when the time comes – in the coding stage.

These "Fusedocs with stuff added to them" are known as fuse stubs. I typically write the Fusedoc in HomeSite+, using the available VTMs for Fusedoc. Then I copy and paste the Fusedoc into the Notes pane for its node in the mind map. I do this so that I have the entire architecture for the application contained in one place – the mind map. A sample fuse stub is shown in Listing 1.

Generate the Framework

With the mind map completed, it was time to start working some time-saving magic. I saved the mind map in text format, which results in an outline with all the data in the mind map. A few years back, I wrote a custom tag called Fuseminder that uses just such a file to generate the entire Fusebox framework. With feedback and new code provided by several other Fuseboxers over the years, the latest being Jamie Thomas, Fuseminder was kept up-to-date as new versions of Fusebox came out. So I ran the latest, FuseminderFB4, passing the outline file to it. The calling form for FuseminderFB4 is shown in Figure 4.

FuseminderFB4 cruises the outline file, creating directories, writing data to fuse stub files, creating circuit configuration files (circuit.xml) and an application configuration file (fusebox.xml), and placing the Fusebox 4 core files in the application's root directory. Figure 5 shows a portion of the output from FuseminderFB4 running in Verbose mode.

Test Harnesses

With the application framework generated, it was time to build some test harnesses for unit testing. Test harnesses are just little templates that set up the environment required by a fuse and allow the coder to run the fuse all by itself. This allows them to make sure the fuse works according to its Fusedoc even though the coder knows nothing about the rest of the application.

As you can imagine, writing all these test harnesses is quite a bit of drudge work. It's another perfect opportunity for the use of a utility, so I wrote one called Harness (back in old days of Fusedoc 1.0). Harness2 is the latest incarnation, updated for Fusedoc 2 by several of us, notably Kevin Roche. I ran Harness2 as a custom tag called in the root of the application, and it cranked out all the test harnesses in a few seconds. Listing 2 shows a test harness for the fuse stub we saw in Listing 1.

Coding

Test harnesses were generated for all the fuses, and the time came to write the actual code for the application. As the architect, I farmed out the coding assignments to my skilled team of coders and waited for the finished fuses to come back. To keep track of the coding assignments, I used another utility I wrote – FB4Checklister. This utility just reads through the application's circuit.xml files and renders a checklist for all the fuseactions and fuses in the each circuit. A page of output from FB4Checklister is shown in Figure 6.

With the fuse stubs, test harnesses, and checklist prepared, I called in a few favors. With Hal unable to attend the MAX conference, I enlisted his help in writing the final application code. With a few circuits in hand, Hal went to work on his fuses while I worked on the rest of them. As of this moment, the fuses need to be written. I'll finish this story after the application is done, sometime during MAX.

Sometime During Max – Integration Testing

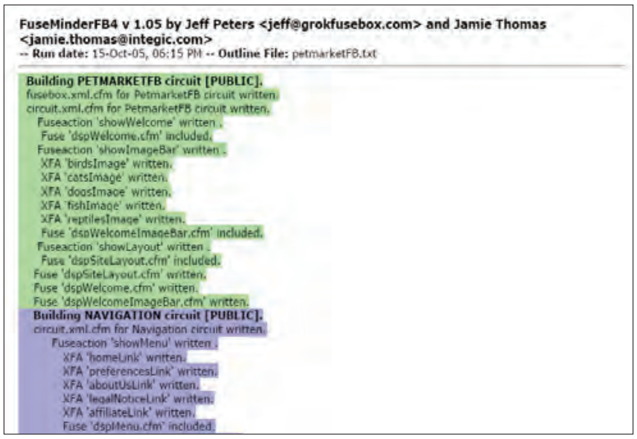
The coders went to work on their fuses, and let me know if they had questions about any of the Fusedocs. After a few clarifications on some fuses, the work was completed and I had the code back in hand. It was time to drop everything into the application framework and watch the finished application run flawlessly.

Okay, so the "run flawlessly" part didn't quite happen. Why? Because I, as the architect, rushed through my job and didn't do a good job designing the application. Consequently, there were a few issues that I had to resolve during the final integration testing. Fortunately, these were all minor problems, such as specifying the variable "productID" instead of "breedID", and easy to resolve thanks to the granular nature of the code.

What's It Look Like?

Obviously, the application itself doesn't differ from the original in terms of the user experience. The important part of the exercise for us developers is how the code looks. One of the nicest things for Fusebox users is the "roadmap" nature of each circuit's configuration file (circuit.xml) and the application configuration file (fusebox.xml). Listing 3 shows the fusebox. xml file for PetmarketFB, and Listing 4 shows the circuit.xml file for the Catalog circuit.

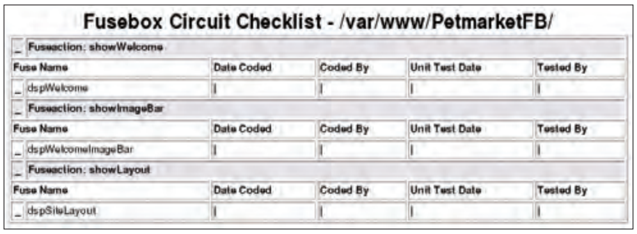
The code in the fuses isn't as critical in terms of differences from other frameworks, but I will make an example of the overall application layout display fuse, shown in Listing 5. It's immediately apparent that this code, and the layout is creates, is very simple to interpret. The actual content blocks have been removed and replaced by Component Content Variables. This is one small example of how Fusebox can greatly simplify the maintenance of an application.



```
FuseMinderFB4 v 1.05 by Jeff Peters <jeff@grokfusebox.com> and Jamie Thomas
<jamie.thomas@integic.com>
-- Run date: 15-Oct-05, 06:15 PM -- Outline File: petmarketFB.txt

Building PETMARKETFB circuit [PUBLIC].
fusebox.xml.cfm for PetmarketFB circuit written.
circuit.xml.cfm for PetmarketFB circuit written.
  Fuseaction 'showWelcome' written .
   Fuse 'dspWelcome.cfm' included.
  Fuseaction 'showImageBar' written .
   XFA 'birdsImage' written.
   XFA 'catsImage' written.
   XFA 'dogsImage' written.
   XFA 'fishImage' written.
   XFA 'reptilesImage' written.
   Fuse 'dspWelcomeImageBar.cfm' included.
  Fuseaction 'showLayout' written .
   Fuse 'dspSiteLayout.cfm' included.
 Fuse 'dspSiteLayout.cfm' written.
 Fuse 'dspWelcome.cfm' written.
 Fuse 'dspWelcomeImageBar.cfm' written.
 Building NAVIGATION circuit [PUBLIC].
 circuit.xml.cfm for Navigation circuit written.
    Fuseaction 'showMenu' written .
      XFA 'homeLink' written.
      XFA 'preferencesLink' written.
      XFA 'aboutUsLink' written.
      XFA 'legalNoticeLink' written.
      XFA 'affiliateLink' written.
      Fuse 'dspMenu.cfm' included.
```

Figure 5: FuseminderFB4 output



Fusebox Circuit Checklist - /var/www/PetmarketFB/

_ Fuseaction: showWelcome				
Fuse Name	Date Coded	Coded By	Unit Test Date	Tested By
_ dspWelcome	|	|	|	|
_ Fuseaction: showImageBar				
Fuse Name	Date Coded	Coded By	Unit Test Date	Tested By
_ dspWelcomeImageBar	|	|	|	|
_ Fuseaction: showLayout				
Fuse Name	Date Coded	Coded By	Unit Test Date	Tested By
_ dspSiteLayout	|	|	|	|

Figure 6: FB4Checklister output

Similarly, the layout fuse for the checkout process is much more flexible, as shown in Listing 6. The layout doesn't depend on an arbitrary "step" variable to control segments of the display. Consequently, this layout can be easily adapted to any stepped process simply by changing the step names contained in the list at the beginning of the fuse. Displayed links in the title bands and the display of content within successive bands is controlled by the presence or absence of content instead of a control variable's value.

Finally

At the end of the FLiP process, we have a finished application that is highly documented, easy to maintain, and has powerful management features built right in. There are a few caveats about this particular project that I should mention. For example, without the original database designer available, assumptions had to be made about the intent of the database design. For example, there appears to be no place in the database to store orders – only user shopping carts. My assumption based on this observation was that the orders were being stored elsewhere in the company. There are also inconsistencies in the database, such as calling a field "firstname" in one table and "givenname" in another. By using the FLiP process, these inconsistencies don't become inconsistencies in the application, because the application's variables, field names, and so on are designed without the database at hand, and the differences in naming are resolved during the coding phase.

In addition, issues like the layout of the site, which is fairly confusing to examine and navigate in its original code, becomes much more easy to follow when organized in terms of Fusebox layout files and Component Content Variables. Fusebox provides the organization that improves the lives of maintenance programmers.

This is a simple introduction to Fusebox and FLiP in the context of the Pet Market application. All the files for the application, captured at various stages of the process, are available in a zip file on my site, www.GrokFusebox.com.

About the Author

Jeff Peters is a program manager and application architect for Operational Technologies Services in Vienna, Virginia. His Cold-Fusion-related books are available at www.protonarts.com.

jeff@grokfusebox.com

Listing 1: Fuse Stub

```
<!---
<fusedoc fuse="qryGetBreed.cfm" language="ColdFusion" specifica-
tion="2.0">
  <responsibilities>
    I return a recordset of data for the specified breed.
  </responsibilities>
  <properties>
    <history author="jeff@grokfusebox.com" role="Architect" date="2005-
10-11" type="Create" />
  </properties>
  <io>
    <in>
      <number name="breedID" precision="Integer" scope="attributes"
optional="No" />
    </in>
    <out>
      <recordset name="qryGetBreed">
        <number name="categoryID" precision="integer" optional="No" />
        <number name="breedID" precision="Integer" />
        <string name="breedName" />
        <string name="breedDescription" />
      </recordset>
    </out>
  </io>
</fusedoc>
--->

<cf_querysim>
qryGetBreed
categoryID,breedID,breedName,breedDescription
1|101|Finch|The little finch provides lots of entertainment, and has a
soft cheerful voice and unique personality. A pleasure to care for,
its soft chirping is soothing and pleasant. They are happy in quiet
surroundings.
</cfquerysim>
```

Listing 2: Test Harness

```
<!-- tst_actGetCategoryTitle.cfm -->
<!--- -->
<fusedoc fuse="tst_actGetCategoryTitle.cfm" language="ColdFusion"
specification="2.0">
<responsibilities>
  I am a unit test harness for the actGetCategoryTitle.cfm fuse.
</responsibilities>
<properties>
  <history date="16/Oct/2005" author="Your Name Here" email="yourname@y
ourdomain.com" type="create"/>
</properties>
</fusedoc>
 --->

<!--- ********************************************************* --->
<!--- The following lines are copied from the .ini file         --->
<!---   [Settings] in C:\CFusionMX7\CustomTags\harness2.ini  --->
<!--- ********************************************************* --->

<cfinclude template="../fusebox.init.cfm">

<!--- The following lines set up the fusebox API environment.   --->
<cfscript>
fusebox=StructNew();
fusebox.IsCustomTag = False;
fusebox.IsHomeCircuit = True;
fusebox.IsTargetCircuit = True;
fusebox.Circuit = "foo";
fusebox.Fuseaction = "";
fusebox.HomeCircuit = "";
fusebox.TargetCircuit = "";
fusebox.ThisCircuit = "";
fusebox.ThisLayoutPath = "";
fusebox.CurrentPath = "";
fusebox.RootPath = "";
</cfscript>

<!--- ********************************************************* --->
<!--- The following lines are generated from the fusedoc.       --->
<!--- ********************************************************* --->

<cfset attributes.categoryID=1>

<!--- ********************************************************* --->
<!--- Now include the fuse to be tested.                        --->
<!--- ********************************************************* --->
<cfinclude template="actGetCategoryTitle.cfm">


<!--- ********************************************************* --->
<!--- The following lines are copied from the .ini file        --->
<!---   [ShowResults] in C:\CFusionMX7\CustomTags\harness2.ini  --->
<!--- ********************************************************* --->

<!--- The following lines are included after the fuse to be tested.
--->
<cfdump var="#variables.categoryTitle#">
```

Listing 3: fusebox.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Auto-generated by FuseminderFB4 for Jeff Peters (jeff@grokfusebox.
com) 15-Oct-05 06:15 PM -->

<fusebox>

  <circuits>

    <circuit alias="PetmarketFB" path="" parent="" />

    <circuit alias="Navigation" path="Navigation/" parent="PetmarketFB"
/>

    <circuit alias="Search" path="Search/" parent="PetmarketFB" />

    <circuit alias="Catalog" path="Catalog/" parent="PetmarketFB" />

    <circuit alias="Users" path="Users/" parent="PetmarketFB" />

    <circuit alias="Information" path="Information/"
parent="PetmarketFB" />

    <circuit alias="Cart" path="Cart/" parent="PetmarketFB" />

  </circuits>


 <parameters>
    <parameter name="fuseactionVariable" value="fuseaction" />
    <parameter name="defaultFuseaction" value="home.main" />
    <parameter name="precedenceFormOrUrl" value="form" />
    <parameter name="mode" value="development" />
    <parameter name="password" value="" />
    <parameter name="parseWithComments" value="true" />
    <parameter name="scriptlanguage" value="cfmx" />
    <parameter name="scriptFileDelimiter" value="cfm"/>
    <parameter name="maskedFileDelimiters" value="htm,cfm,cfml,php,php
4,asp,aspx" />
```

```
    <parameter name="characterEncoding" value="utf-8" />
  </parameters>

  <globalfuseactions>
    <preprocess>
 </preprocess>
    <postprocess>
 </postprocess>
  </globalfuseactions>

  <plugins>
    <phase name="preProcess">
    </phase>
    <phase name="preFuseaction">
    </phase>
    <phase name="postFuseaction">
    </phase>
    <phase name="fuseactionException">
    </phase>
    <phase name="postProcess">
    </phase>
    <phase name="processError">
    </phase>
  </plugins>
</fusebox>
```

Listing 4: circuit.xml

```
<!-- Auto-generated by FuseminderFB4 for Jeff Peters (jeff@grokfusebox.
com) 15-Oct-05 06:15 PM -->
<circuit access="public">
  <prefuseaction>
  </prefuseaction>


  <fuseaction name="showBreed">

    <include template="qryGetBreed" />

    <include template="qryBreedItems" />

    <include template="actGetCategoryTitle" />

    <xfa name="addToCartButton" value= "cart.addItemsToCart" />

    <include template="dspBreed" />

  </fuseaction>


  <fuseaction name="showCategoryBreeds">

    <include template="qryBreedNames" />

    <xfa name="breedLink" value= "catalog.showBreed" />

    <include template="dspCategoryBreeds" />

  </fuseaction>


  <postfuseaction>
  </postfuseaction>
</circuit>
```

Listing 5: Layout Fuse

```
<!---
<fusedoc fuse="dspSiteLayout.cfm" language="ColdFusion" specifica-
tion="2.0">
  <responsibilities>
    I display the final site layout.
  </responsibilities>
  <properties>
    <history author="jeff@grokfusebox.com" role="Architect" date="2005-
10-11" type="Create" />
  </properties>
  <io>
    <in>
      <string name="mainContent" scope="variables" optional="No" />
      <string name="imageBarContent" scope="variables" optional="No" />
      <string name="drillBoxContent" scope="variables" optional="No" />
      <string name="menuContent" scope="variables" optional="No" />
      <string name="searchBarContent" scope="variables" optional="No"
/>
      <string name="adContent" scope="variables" optional="No" />
    </in>
    <out>
    </out>
  </io>
</fusedoc>
--->

<cfparam name="mainContent" default="">
<cfparam name="imageBarContent" default="">
<cfparam name="drillBoxContent" default="">
<cfparam name="menuContent" default="">
<cfparam name="searchBarContent" default="">
<cfparam name="adContent" default="">

<cfoutput>
<html>
<head>
  <title>Pet Market</title>
  <style>
    b{font-family:arial,geneva,helvetica,sans-serif;}
    a:hover{color:009900;text-decoration:underline;}
    a.button:hover{color:00ff00;}
    .button:visited{color:00aa00;}
    td {font-family:arial,geneva,helvetica,sans-serif;}
    .text{font-family:arial,geneva,helvetica,sans-serif;font-size:75%;
line-height:135%;}
    .label{font-family:arial,geneva,helvetica,sans-serif;font-
size:70%;}
    .labelstrong{font-size:70%;color:##000000;font-weight:bold;}
    .headline{font-family:arial,geneva,helvetica,sans-serif;font-
size:125%;font-weight:bold;}
  </style>
</head>
<body  background="images/dog_bg_1800.jpg" leftmargin="0" topmargin="0"
style="background-repeat: no-repeat;" link="336633" vlink="336633"
bgcolor="white">
  #menuContent#
  <table border="0" cellpadding="0" cellspacing="0" width="800">
    <tr valign="top">
      <td width="175" nowrap align="right">
        <img src="images/clear.gif" alt="" width="1" height="1"
hspace="85" vspace="85" border="0" alt=" "><br clear="all">
        #drillBoxContent#
      </td>
      <td>
        <img src="images/clear.gif" alt="" width="1" height="1"
vspace="25" border="0" alt=" "><br clear="all">
        <table border="0" cellpadding="0" cellspacing="0"
bgcolor="black" width="400">
          <tr>
            <td>
              <table border="0" cellpadding="0" cellspacing="1"
```

SiteExecutive^{tm} Web Content Management

Empower your users to take control of the web

Hundreds of web sites and thousands of people use Site**Executive** 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

SYSTEMS**ALLIANCE**
Think Big. Work Smart

Sun microsystems Partner Advantage

Consolidate your infrastructure and increase performance by leveraging the SunFire^{tm} x64 Server Family, powered by multi-core or single-core AMD Opteron^{tm} processors running Solaris^{tm} 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.

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	48
COMMUNITYMX	WWW.COMMUNITYMX.COM/TRIAL		51
ECLIPSE	HTTP://ECLIPSE.SYS-CON.COM	888-303-5282	41
HOSTING.COM	WWW.HOSTING.COM		21
HOSTMYSITE	WWW.HOSTMYSITE.COM	877-215-4678	27
INTERAKT	WWW.INTERAKTONLINE.COM/	4031-401-68-19	3
INTERMEDIA	WWW.INTERMEDIA.NET	888-379-7729	52
MACROMEDIA	MACROMEDIA.COM/GO/CFMX7_DEMO.	415-252-2000	2
MACROMEDIA	WWW.MACROMEDIA.COM/GO/8_STUDIO8	415-252-2000	13
PAPERTHIN	WWW.PAPERTHIN.COM	800-940-3087	25
SYS-CON WEBSITE	WWW.SYS-CON.COM	201-802-3000	35
SYSTEMS ALLIANCE	WWW.SITEEXCUTIVE.COM/CFDJ	877-797-2554	33
VERIO	WWW.VERIO.COM/PARTNERS	866-237-4121	23
VITALSTREAM	WWW.VITALSTREAM.COM	800-254-7554	19
WEBAPPCABARET	WWW.WEBAPPCABARET.COM/JDJ.JSP	866-256-7973	6
WEBAPPER	WWW.SEEFUSION.COM		47
WSE/OPEN SOURCE	WWW2.SYS-CON.COM/EVENTS	201-802-3066	39

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

```
bgcolor="black" width="100%">
                <tr>
                  <td>
                    #imageBarContent#
                  </td>
                </tr>
                <tr valign="top">
                  <td>
                    #mainContent#
                  </td>
                </tr>
              </table>
            </td>
          </tr>
        </table>
      </td>
      <td width="100%">
        <table border="0" cellpadding="0" cellspacing="0">
          <tr>
            <td height="65"></td>
          </tr>
        </table>
        #searchBarContent#
        #adContent#
      </td>
    </tr>
  </table>
  <br><br>
</body>
</html>

</cfoutput>
```

Listing 6: Checkout Layout Fuse

```
<!---
<fusedoc fuse="dspCheckoutLayout.cfm" language="ColdFusion" specifica-
tion="2.0">
  <responsibilities>
    I display the component variables that make up the shopping check-
out.
    If the content of a particular band is empty, I put a link to the
band's
    XFA around the band's title.  Otherwise, the band's title doesn't
have a link.
  </responsibilities>
  <properties>
    <history author="jeff@grokfusebox.com" role="Architect" date="2005-
10-11" type="Create" />
  </properties>
  <io>
    <in>
      <string name="xfa.loginLink" optional="No" />
      <string name="xfa.customerDetailsLink" optional="No" />
      <string name="xfa.shippingAddressLink" optional="No" />
      <string name="xfa.shippingOptionsLink" optional="No" />
      <string name="xfa.paymentMethodLink" optional="No" />
      <string name="checkoutLoginBand" scope="variables" optional="Yes"
default="" />
      <string name="checkoutCustomerDetailsBand" scope="variables"
optional="Yes" default="" />
      <string name="checkoutShippingAddressBand" scope="variables"
optional="Yes" default="" />
      <string name="checkoutShippingOptionsBand" scope="variables"
optional="Yes" default="" />
      <string name="checkoutPaymentMethodBand" scope="variables"
optional="Yes" default="" />
      <string name="checkoutConfirmationBand" scope="variables"
optional="Yes" default="" />
      <string name="checkoutReceiptBand" scope="variables"
optional="Yes" default="" />
    </in>
    <out>
    </out>
  </io>
</fusedoc>
--->
<cfset bandsList = "Login,CustomerDetails,ShippingAddress,ShippingOptio
ns,PaymentMethod,Confirmation,Receipt">

<cfset bandContentMask = 0>
<cfloop list="#bandsList#" index="thisBand">
  <cfparam name="checkout#thisBand#Band" default="">
  <cfset thisBandPower = (ListFind(bandsList,thisBand) - 1)>
  <!--- Add a flag to the bitmask if this band has content --->
  <cfif Variables['checkout' & thisBand & 'Band'] GT "">
    <cfset bandContentMask = bandContentMask + (2^thisBandPower)>
  </cfif>
</cfloop>

<cfset gotFocusColor = "ffee99">
<cfset notFocusColor = "ddddcc">


<cfoutput>
<table width="100%">

<cfloop list="#bandsList#" index="thisBand">
  <cfset thisBandIndex = ListFind(bandsList,thisBand)>
  <cfset thisBandPower = (ListFind(bandsList,thisBand) - 1)>
  <cfset "#thisBand#LinkOpen" = "">
  <cfset "#thisBand#LinkClose" = "">
  <!--- If any bands greater than this one have content, this one's
header needs a link --->
  <cfif (bandContentMask GT 2^thisBandPower) AND (thisBandIndex LT
ListLen(bandsList) - 1)>
    <cfset "#thisBand#LinkOpen" = "<a href=""#request.
self#?#application.fusebox.fuseactionVariable#=#xfa[thisBand &
'Link']#"">">
    <cfset "#thisBand#LinkClose" = "</a>">
  </cfif>
  <cfif Variables['checkout' & thisBand & 'Band'] GT "">
    <cfset "#thisBand#BG" = gotFocusColor>
  <cfelse>
    <cfset "#thisBand#BG" = notFocusColor>
  </cfif>
  <tr bgcolor="#variables[thisBand & 'BG']#">
    <td height="18">&nbsp;
      <b class="labelstrong">#variables[thisBand & 'LinkOpen']##thisBan
dIndex#) #thisBand##variables[thisBand & 'LinkClose']#</b>
    </td>
  </tr>
  <cfoutput>#variables['checkout' & thisBand & 'Band']#</cfoutput>
</cfloop>

</table>
</cfoutput>
```

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

Reengineering the CF Pet Market with SAM

A more sensible code base

By Simon Horwith

I've been fairly outspoken for quite some time now about the fact that I don't prescribe to any framework. I've also spent many years refining what's proven to be the best methodology possible for developing ColdFusion applications. I recently promised my boss that I would learn and evaluate several of the popular ColdFusion frameworks.

If I couldn't find one that I felt good about recommending as our standard platform for development, I would formally document the methodology that I use for designing and developing applications. After evaluating several popular frameworks and coming to the conclusion that no one framework would ever meet all of my requirements, I went a step further and began promising to publicly release my methodology.

When I decided that we should have an issue of *CFDJ* entirely committed to showcasing the same application implemented in each of several popular frameworks, I knew that it would be the ideal opportunity to show developers what an application built using my methodology might look like. There's been some speculation about this methodology, which people have been referring to simply as "Simon's methodology." Some people have even made the assumption that it's actually just a "framework in methodology's clothes" and not a methodology at all, which is interesting given the fact that to date there's been no documentation about it whatsoever. I'm still not quite ready to release the formal documentation, but I will publish it online very soon and I will also be releasing a book on software architecture that is a detailed teaching of this methodology. I am, however, happy to publicly give a name to this methodology for the first time… "SAM."

SAM stands for "Sensible Assembly Methodology." I named it this because it is a methodology; one of its main objectives is to build applications by assembling standalone APIs; and it results in a code base that is, in my opinion, easier to follow, maintain, and extend… in other words, a more sensible code base. I had to resist the temptation to call it "Sensible ColdFusion Assembly Methodology" because as funny an acronym that I thought "SCAM' would be, I thought it'd be best if I tried to avoid tempting any naysayers too much. SAM is a combination of the ideas and techniques that I have found most useful in the real world – ideas that are borrowed from object oriented-programming, aspect-oriented programming, streamlined object modeling, and personal experience. Rather than just write about the rules that govern this methodology, I've re-architected the CF Pet Market application much the way it would look were I building it from scratch using SAM, and will explain some of the core ideas behind the methodology while using this CF Pet Market code for illustration. Those of you who are interested in learning more about the methodology should keep an eye on my blog (http://www.horwith.com) – I will be announcing the availability of the SAM documentation there first.

The primary objective of application development with SAM is to create a code base that is portable and easy to follow, maintain, and extend. This idea does not just apply to the application as a whole, but to all of its individual parts as well. This is achieved by developing each aspect and domain object as an API – a standalone module with an easy-to-use interface. A typical API consists of two files: an object (ColdFusion Component) and a custom tag. Only custom tags interact with CFCs, and only CFCs perform server-side business logic. This results in browsable pages that usually consist of little more than a few custom tag calls. This approach ensures the separation of client-side controller logic and presentation tier code from business logic, makes the CFCs much easier to use since a developer need only call custom tags to leverage application objects, and results in well-modularized code that is very reusable, including from one application to the next. Unlike frameworks, SAM has no core files, but it is likely that each developer who uses SAM will have a personal library of APIs and files that they reuse from one application to the next.

Every application built with this methodology is object modeled first, then implemented in code. SAM has a well-defined development process, very similar to FLIP (Fusebox Lifecycle Process), but describing this is beyond the of scope of this article as is object modeling; however, for those of you familiar with UML, think of an object model as being very similar to a UML class diagram. SAM uses a school of thought on object modeling applications called "object think" – it's a set of rules that govern how to represent entities and assign responsibility to objects. Let's look at the code in order to see how some of these ideas are put into practice.

The original application created an "extensions" directory above the wwwroot. This is fine but I prefer, at least in development, to keep all of my code beneath the Web root – this is simply more portable. The "extensions" directory has a "customtags' and a "components" subdirectory. There's one file in "customtags" and the "components" directory contains only a "petmarket" subdirectory (with five CFCs and a "_product" subdirectory containing one .cfm file). You can see the entire directory tree created by unzipping the original application shown in Figure 1. To make things more portable, I created a "customtag" directory and a "cfc' directory beneath my wwwroot directory. The CFC directory has no subdirectories and contains only ColdFusion Components, and the customtag directory contains only custom tags. I also created a "css" directory because moving the styles into their own files in their own directory structure makes the code easier to maintain. The original application had all styles defined in a <STYLE> block inside of the default page wrapper tag (I moved this block into a CSS file, which I put in my "css" directory). The directory structure for the SAM version is shown in Figure 2. Notice that there are now six directories as opposed to the original eight, and that everything other than the "db" directory (which has only database files) is nested beneath the "wwwroot". Now let's examine the files that are in the SAM version and compare them with the original application.

I stated earlier that with SAM, object model the application first. Figure 3 shows the CFC files that were in the original application's "components/petmarket" directory. Figure 4 shows the CFC files in the "cfc" directory for the SAM version. In the SAM rewrite, I have added a "search" component, an "objectFactory" component, and a "dataAccess" component. Both versions have a "user" object, a "cart" object, and a "product" object, and the SAM version has a "catalog" and a 'baseObject" object but the original version has a "_base" and a "category" object. You may be asking yourself why, if SAM makes things easier, there are more CFCs?

The object factory object was added in order to simplify the retrieval of objects throughout the application. A single CFC in the application scope now gives all of the code in the application access to whatever object they require. Different objects are handled differently by the factory – for example, some are singletons whereas others are created on the fly. I already had a copy of an object Factory CFC since I use them a lot, so I copied it into the CFC folder and modified it for the Pet Market objects. The data access component houses methods for storing/retrieving data that is not specific to any one object in my business domain (you could think of it as global data) – for the Pet Market it only needs to store two simple queries (one for states and one for shipping methods). Again, I already had a data access template file that I copied and customized. The search object is yet another CFC that I already had an API built for – along with its tag I can easily add consistent search functionality, navigation, and results display into any application.

The "baseObject" is another CFC I had already written: I use some variation of this object in nearly every application that I build. The original application "_base" object really didn't much at all – it contained one method used by the application, which simply looped over

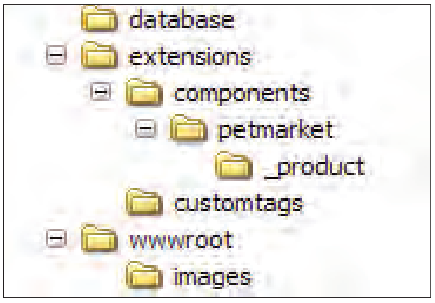


Figure 1

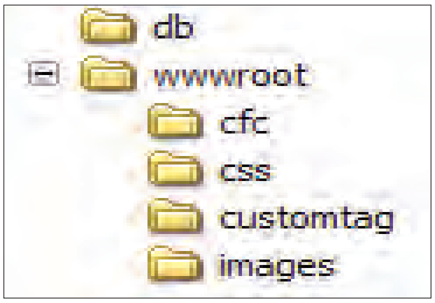


Figure 2

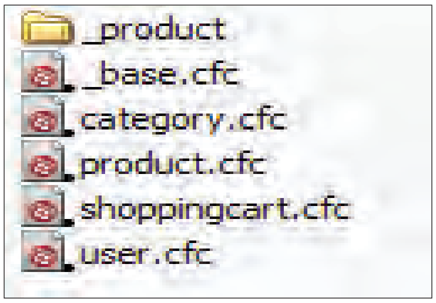


Figure 3

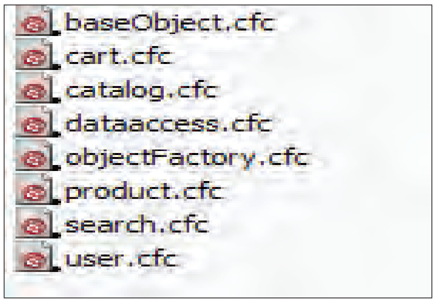
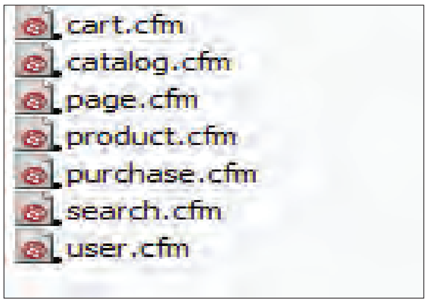


Figure 4

the “this” scope in order to compile and return a structure of all the properties for any object that inherits from it. In SAM, data is kept as private as possible within objects (or tags or anything else, for that matter), so no data is stored in the “this” scope. All domain objects extend my “baseObject”, which has generic property accessors for getting and setting. My baseObject also includes functionality for easily defining what properties may be set or retrieved, for defining validation rules on properties, for validating property values, and for retrieving validation error information.

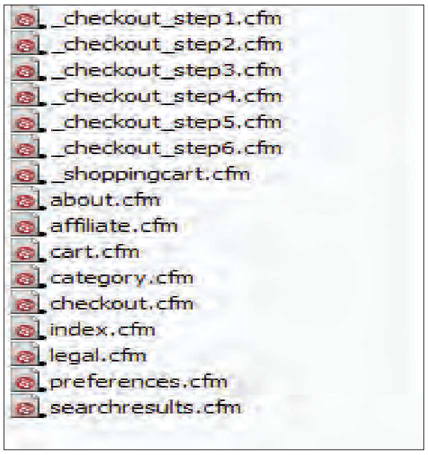
Another major difference between the two CFC code bases is that the original application had a “category” object whereas the SAM code has a “catalog” object. In “object think” terms, a catalog is made up of products and the products fall into categories (which are really just metadata), and a user owns a shopping cart and a search object (searches are user specific). In the original version there is no clear ownership of data or functionality, but these are clearly defined in the rewrite. The original application had some CFC methods that output directly to the screen, but this is not allowed when following the Sensible Assembly Methodology. The impact of writing to the screen from tag APIs rather than objects is clear when comparing the number of tags and browsable files in the two applications.

The custom tags folder in the original application contained a single tag, “wrapper.cfm”, which basically creates the look and feel of the site (header, footer, navigation, etc.). Each browsable page wraps a call to the wrapper tag around whatever it wants to display, and the tag calls a few CFC methods and performs a few includes. Many of the browsable files in the original application also use includes between the opening and closing calls to “wrapper” in order to generate their output. Figure 5 shows the contents of the “customtag” directory in my SAM application, which contains not one but seven custom tags! This is because of what we discussed earlier in the article – all objects have an accompanying custom tag that makes the object functionality and its different views easy to invoke via a tag-syntax interface. It’s fairly obvious which objects are used by each of the custom tags simply by looking at the tag names; in fact, all of the tag names directly correspond with the name of a CFC, save one. The “purchase” tag facilitates the flow of a multi-step checkout process when the user wants to check out the shopping cart. In more complex applications, I oftentimes do have a purchase object, but it was overkill for this simple application and limited timeframe. Creating all of these tags took very little time – I already had the search and user tags, and most of the work for all of the tags was nothing more than copying code from the original include files, adding some CFC interaction and instantiation, and putting it all into a nice format.



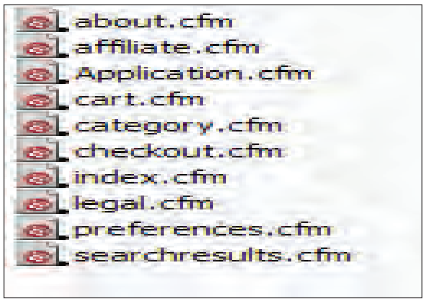
```
cart.cfm
catalog.cfm
page.cfm
product.cfm
purchase.cfm
search.cfm
user.cfm
```

Figure 5



```
_checkout_step1.cfm
_checkout_step2.cfm
_checkout_step3.cfm
_checkout_step4.cfm
_checkout_step5.cfm
_checkout_step6.cfm
_shoppingcart.cfm
about.cfm
affiliate.cfm
cart.cfm
category.cfm
checkout.cfm
index.cfm
legal.cfm
preferences.cfm
searchresults.cfm
```

Figure 6



```
about.cfm
affiliate.cfm
Application.cfm
cart.cfm
category.cfm
checkout.cfm
index.cfm
legal.cfm
preferences.cfm
searchresults.cfm
```

Figure 7

Figure 6 shows the cfm files that are in the original Web root. There are 16 files in all – each one is either browsed directly or included by another file. Figure 7 shows the files in the SAM Web root. There are 10 files in this version and, other than the Application.cfm file, these files only represent the files that are directly browsed by users. There are no includes and no CFC instantiation or interaction in any of these files, and all display code is generated by custom tags. One result of this technique is that the browsable files are tiny. The original site has quite a bit of conditional logic and formatting in the browsable pages, whereas the browsable pages in the SAM application consist of absolutely nothing but calls to custom tags and nested plain text for the simple pages that need nothing more than the default look and feel and some centered text.

The power of the Sensible Assembly Methodology’s use of APIs is illustrated nicely by the code in the Pet Market checkout process. In the original application, one file contains all of the business logic to process data submitted in seven steps and to facilitate the flow between these steps, which includes performing a <CFINCLUDE> for the display component of each step. With SAM, the same file consisted of a call to the page wrapper and a nested call to the purchase tag. Doesn’t this just mean that I moved all of that ugly code into a custom tag? No. In fact, it doesn’t mean this at all. The purchase tag doesn’t do much more than wrap the appropriate look and feel around the view for each step. The view for each of the seven steps (aside from the final “success message”) is generated by a call to some other custom tag – whether summary data or forms. Each step in the checkout process (each custom tag) bears the responsibility for processing its own data and telling the purchase tag to display

ENGAGE AND EXPLORE...

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

CALL FOR PAPERS NOW OPEN!

SOA WebServices Edge 10th International conference+expo Edge 06

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

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 Developer's Journal, eclipse) developer's journal, WebSphere Journal, Information Storage+Security Journal, wldj, JDJ, LinuxWorld Magazine, MX developer's journal, asp.netPRO, SD Times, CoDe, Software Test & Performance

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

Attention Exhibitors:

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

For Exhibit and Sponsorship Information ► Call 201 802-3023

Produced by SYS-CON EVENTS

the appropriate next step. It's a much better example of code reuse and a clean separation of the flow of control from the view and business logic.

In general, I felt that the CF Pet Market wasn't the best sample application to re-implement within a framework or methodology because it is both simple and has some odd quirks and inconsistencies. For example, some objects have numeric auto-incrementing IDs in the database and some have strings (like UUIDs or locales) for primary keys. Some functionality should be there but isn't (like an admin area, for example), and some functionality cannot be supported by the database as-is (preferences are stored in the session but can never be committed to the database, for example). This isn't a big deal, but it's frustrating when you are the developer who has to work on it. On the other hand, most developers are only willing to spend a limited amount of time developing a sample application or looking at another developer's sample application, so perhaps the simplicity of the Pet Market application is a blessing. When implementing my SAM version of CF Pet Market, I was careful to leave all of the existing functionality exactly as it is in the original. By that I mean that I didn't set out with the goal of modifying, enhancing, or adding to the existing functionality. It's an exact replica of the original application from an end-user point of view, an exact replica with a much more portable code base that's easier to read and maintain.

Building an application as a conglomerate of APIs makes it very easy to reuse code within an application. Were the owner of the pet market site to come to me tomorrow and ask me to add an administrative back end for managing all of the data, it would take no time at all – I just need to create pages that call the custom tags and pass them the appropriate data. This reusability also means that I can take each of these components individually and use them in other applications. If my next project needs a shopping cart, I have a cart and tag all ready to go. In fact, the "cart," "user," "search," and "product" objects and tags in this application were all added by me copying files that I already had from other applications. All I had to do was customize them a little for this app. I had to resist leaving in extra functionality that would have made it too robust – for example, the user object that I began with actually had all of the functionality required for roles and groups based authentication and authorization. It was beyond the scope of these requirements so I removed all of that functionality. I had to make the object factory "requestObject()" method accept a string argument rather than numeric because, though my applications always have numeric IDs in the database, this one didn't. SAM applications are usually aware of "context" – what part of the application the current request is being made within. Again, this was overkill for the Pet Market application so I removed this feature. This is one of the things I like about methodologies as opposed to frameworks – because a methodology has no core files, the developer is encouraged to modify all of the files to suit his or her needs. Having generic versions of objects, and tags to work with those objects that you can quickly and easily integrate with any application, is how developers who use SAM are still able to achieve quick results without having a framework already in place.

I prefer to use the Sensible Assembly Methodology because the applications are easier to maintain and extend, because the skills and code are transferable anywhere, and because it doesn't inhibit or hide anything from the developer while encouraging best practices. Perhaps what I like more than anything, though, isn't the idea of SAM as an alternative to using frameworks, but the fact that the modules developed with SAM are so portable that they should be able to integrate with any decent framework with little effort from the developer. There is still so much about the Sensible Assembly Methodology that I have yet to document or explain, but I strongly suggest taking a look at the code at www.cfpetmarket.com for my sample application as one good place to start.

simon@horwith.com

WHY 'ECLIPSE DEVELOPER'S JOURNAL' AND WHY NOW? PG 2

eclipse) developer's journal

PREMIER 2005 VOL 1 ISSUE 1

http://eclipse.sys-con.com

What's New in Eclipse 3.1

Polishing the apple ...4

PLUS...

WEB TOOLS PLATFORM: J2EE DEVELOPMENT THE ECLIPSE WAY ...12

'BEING A BETTER PLATFORM' ...22
INTERVIEW WITH MIKE MILINKOVICH, EXECUTIVE DIRECTOR, ECLIPSE FOUNDATION

MYECLIPSE 4.0 UP ON DECK: EXCLUSIVE INTERVIEW WITH MAHER MASRI, PRESIDENT, GENUITEC ...36

ECLIPSE IDE FOR STUDENTS: USEFUL TIPS AND TRICKS ...28

ColdFusion

For more information go to...

U.S.

Alabama
Huntsville
Huntsville, AL CFUG
www.nacfug.com

Alaska
Anchorage
Alaska Macromedia User Group
www.akmmug.org

Arizona
Phoenix
www.azcfug.org

Arizona
Tucson
www.tucsoncfug.org

California
San Francisco
Bay Area CFUG
www.bacfug.net

California
Riverside
Inland Empire CFUG
www.sccfug.org

California
EL Segundo
Los Angeles CFUG
www.sccfug.org

California
Irvine
Orange County CFUG
www.sccfug.org

California
Davis
Sacramento, CA CFUG
www.saccfug.org

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

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

California
Long Beach
Southern California CFUG
www.sccfug.org

Colorado
Denver
Denver CFUG
www.denvercfug.org/

Delaware
Kennett Square
Wilmington CFUG
www.bvcfug.org/

Delaware
Laurel
Delmarva CFUG
www.delmarva-cfug.org

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

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

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

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

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

Georgia
Atlanta
Atlanta, GA CFUG
www.acfug.org

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

Indiana
Avon
Indianapolis, IN CFUG
www.hoosierfusion.com

Indiana
Mishawaka
Northern Indiana CFUG
www.ninmug.org

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

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

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

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

Maryland
Rockville
Maryland CFUG
www.cfug-md.org

Massachusetts
Quincy
Boston, MA CFUG
www.bostoncfug.com

Michigan
East Lansing
Mid Michigan CFUG
www.coldfusion.org/pages/index.cfm

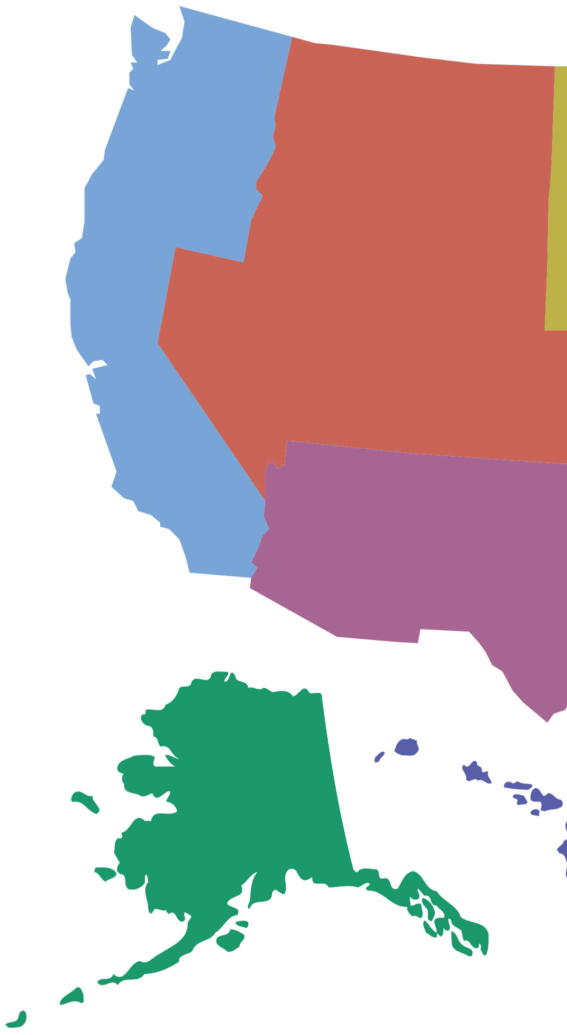
Minnesota
Brooklyn Park
Twin Cities CFUG
www.colderfusion.com

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

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

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

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



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

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

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

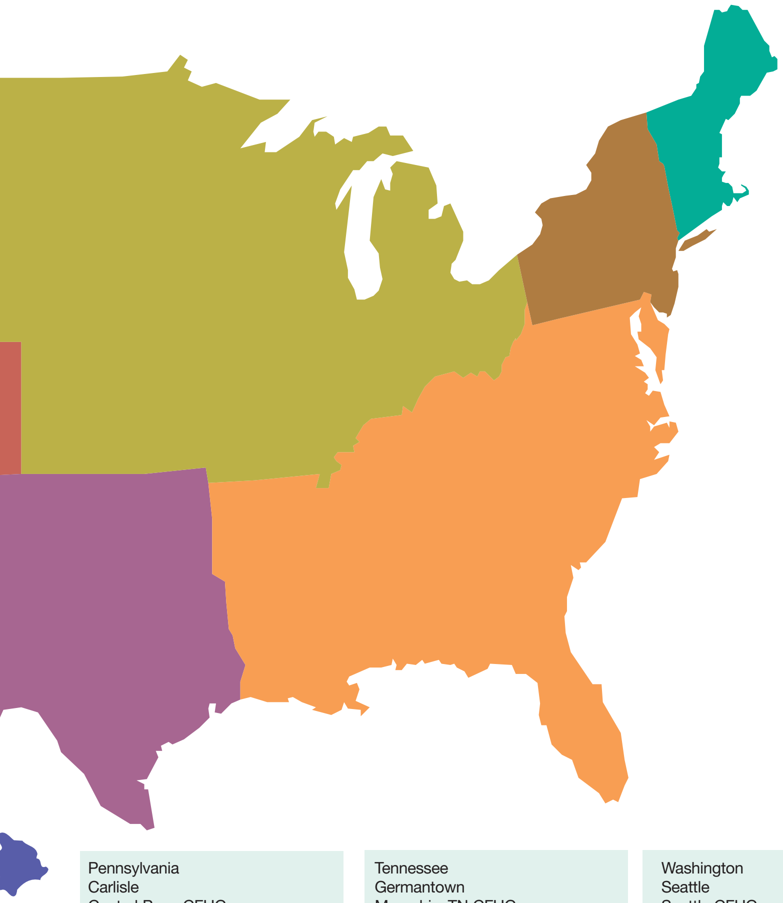
North Carolina
Raleigh
Raleigh, NC CFUG
www.ccfug.org

Ohio
Dayton
Greater Dayton CFUG
www.cfdayton.com

Oregon
Portland
Portland, OR CFUG
www.pdxcfug.org

Jser Groups

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

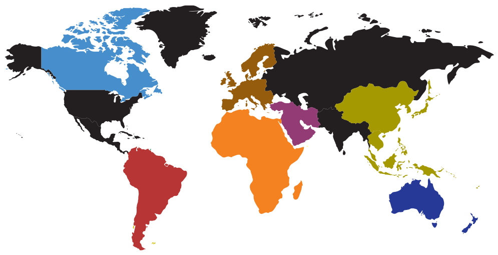


Pennsylvania Carlisle Central Penn CFUG www.centralpenncfug.org	Tennessee Germantown Memphis, TN CFUG www.mmug.mind-over-data.com	Washington Seattle Seattle CFUG www.seattlecfug.com
Pennsylvania Exton Philadelphia, PA CFUG www.phillycfug.org/	**Texas** Austin Austin, TX CFUG www.cftexas.net/	
Pennsylvania State College State College, PA CFUG www.mmug-sc.org/	**Texas** Corinth Dallas, TX CFUG www.dfwcfug.org/	
Rhode Island Providence Providence, RI CFUG www.ricfug.com/www/meetings.cfm	**Texas** Houston Houston Area CFUG www.houcfug.org	
Tennessee LaVergne Nashville, TN CFUG www.ncfug.com	Utah North Salt Lake Salt Lake City, UT CFUG www.slcfug.org	

About CFUGs

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

INTERNATIONAL



Australia ACT CFUG www.actcfug.com	**Italy** Italy CFUG www.cfmentor.com
Australia Queensland CFUG www.qld.cfug.org.au/	**Japan** Japan CFUG cfusion.itfrontier.co.jp/jcfug/jcfug.cfm
Australia Southern Australia CFUG www.cfug.org.au/	**Scotland** Scottish CFUG www.scottishcfug.com
Australia Victoria CFUG www.cfcentral.com.au	**South Africa** Joe-Burg, South Africa CFUG www.mmug.co.za
Australia Western Australia CFUG www.cfugwa.com/	**South Africa** Cape Town, South Africa CFUG www.mmug.co.za
Brazil Brasilia CFUG www.cfugdf.com.br	**Spain** Spanish CFUG www.cfugspain.org
Brazil Rio de Janerio CFUG www.cfugrio.com.br/	**Switzerland** Swiss CFUG www.swisscfug.org
Brazil Sao Paulo CFUG www.cfugsp.com.br	**Thailand** Bangkok, Thailand CFUG thaicfug.tei.or.th/
Canada Kingston, ON CFUG www.kcfug.org	**Turkey** Turkey CFUG www.cftr.net
Canada Toronto, ON CFUG www.cfugtoronto.org	**United Kingdom** UK CFUG www.ukcfug.org
Ireland Dublin, Ireland CFUG www.mmug-dublin.com/	

Pets onTap

Using the onTap framework to reinvent the Pet Market application

By Isaac Dealey

I must admit to having been excited at the prospect of the Pet Market frameworks project when Simon proposed it to us at the Fusebox & Frameworks Conference in September. I once tried to do something similar by creating a small blog application using the three popular frameworks that I was aware of at the time (Fusebox 3-4 and Mach-II) and the onTap framework.

Unfortunately I did all the work myself and didn't have a great venue like the ColdFusion Developer's Journal to publish the results. In the end I think my efforts at a useful comparison of frameworks fell short of what I'd hoped. Naturally I'm excited to see Simon take the initiative to enlist the aid of various framework authors and publish the results in the journal. Although I personally lament the fact that the Pet Market application isn't perhaps an ideal medium to demonstrate the unique strengths or advantages of some frameworks (not only the onTap framework, although in my case display tools in particular spring to mind), I do believe the Pet Market application provides probably the best neutral ground for comparison.

My own conversion of the Pet Market application took a bit longer than I expected. At first I thought it was good that the Pet Market application was designed purely for demonstration purposes and wouldn't be used in a production environment.

On further reflection I wonder if perhaps the opposite is true – that the original application is worse for its attempts to educate programmers new to ColdFusion because anyone using it as a learning tool is apt to learn some bad habits. Here are a few examples of what I mean: no primary keys or foreign key constraints in the database containing many related tables (no enforcement of referential integrity); display code executed in a domain-model CFC; and serpentine multi-step form templates containing both form display and post logic in the same file. I'll omit my complaints about the way in which multiple historical postal addresses are stored. Having said all that, here's an application ripe for the kind of structure any good framework will provide.

A Statement of Benefits

Here's a brief list of the benefits I've found using the onTap framework to reinvent the Pet Market application in no particular order.

- Eliminated code while simultaneously adding functionality
- Replaced ad hoc queries with Object Relational Mapping (ORM) and vendor-agnostic SQL abstraction
- Convenient caching of domain-model objects (products, categories, etc.)
- Products in the shopping cart are locale-specific, dictionary-order sorted (i18n)
- All files located in a single parent directory (no CustomTags directory or other external resources)
- Elimination of the custom tag for the application's layout (individual content templates don't reference the layout tag/template – decreases coupling of the content and format)
- Automated branding of images (home page graphics differ from the remaining site)

Due to time constraints and project requests from Simon, I was unable to implement any of the XHTML or form-management and validation features of the framework and as a result can't list them as benefits in using the framework in this instance although under normal circumstances I would have. The onTap framework's form tools would have replaced a fair amount of custom form validation code in this application in a manner that provides identical client-side and server-side validation with very little coding on my part. The XHTML features would have also provided a more modular method of generating the display, allowing anyone who might license my application to easily modify the display without editing any of my code, eliminating headaches related to later upgrades of my original software.

A Place for Everything and Everything in its Place

Because the onTap framework is designed to take advantage of directory structures, getting started with the migration of Pet Market to this framework is easy. First I created a /petmarket/ directory under my framework application root directory in

Subscribe Today!

SAVE 16%

12 Issues for $89.99

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

SPECIAL ISSUE: COLDFUSION SECURITY

COLDFUSION Developer's Journal

ColdFusionJournal.com

September 2004 Volume: 6 Issue: 9

SANDBOX SECURITY By Alex Hübner page 8

Security: ColdFusion Security Best Practices Knowing the security risks are there is half the battle — Bryan Murphy 16

Security: Public-Key Encryption Making strong encryption nearly painless — Wayne Graham 20

cfcUnit: ColdFusion Unit Testing Framework Paul Kenney's version — Harry Klein 30

<bf> on< cf>: Defending ColdFusion Against... — Ben Forta 36

Real-World Uses: Using ColdFusion for Network Monitoring Find the exact piece of data you need — Ryan Emerle, Matthew I. Fusfield 42

Security: Top Ten Web Security Tips You can't be too careful — Michael Smith 48

Web Services Edge 2005 East February 15-17, 2005 Hynes Convention Center Boston, MA See page 39 for details

Editorial Reaching Deeper Simon Horwith page 5

Community Focus Problems Securing Your Applications? Simon Horwith page 7

CF101 ColdFusion's Application Framework Jeffry Houser page 28

CFUGs ColdFusion User Groups page 40

FOUNDATIONS 'Selling' ColdFusion Hal Helms page 46

RETAILERS PLEASE DISPLAY UNTIL NOVEMBER 30, 2004 $9.99US $9.99CAN

SYS-CON MEDIA

AN ADVANCE LOOK AT BLAC

COLDFUSION

ColdFusionJournal.com

Leveragin or Just F

Editorial Blackstone Robert Diamond page 5

CF Community Tales from the List Simon Horwith page 7

cf101 What's the Best Approach to Software Development? Jeffry Houser page 32

CFUGs ColdFusion User Groups page 42

Training: Fast Track to Fl Macromedia Training 'Flexes' I

CFML: Harnessing the P SQL Server Using Store Use of stored procedures wi architecture in your CFML

Foundations: A We A perfect marriage betw

<BF> on <CF>: C and Data Abstra basic object function

PDFs: Creatin Your CF Appl only functional,

RETAILERS PLEASE DISPLAY UNTIL SEPTEMBER 30, 2004 $9.99US $9.99CAN

SYS-CON MEDIA

- 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

which the Pet Market sub-application would execute. In this directory I put a copy of all the ColdFusion templates in the original application's /wwwroot/ directory. I could have put these templates in the framework root directory, although I didn't want to create a separate copy of the onTap framework core (3MB) for only the Pet Market application, so I created the subdirectory so it could execute as a sub-application under my existing framework installation. Then I copied the new /petmarket/ directory also into the framework /_components/ directory to mirror the root directory. Files in the root directory will be directly accessible to browsers, while files in the /_components/ directory won't (they're protected by an Application.cfm template). Once these files are copied, I replace the contents of each of the templates in the root /petmarket/ directory with a one-line cfinclude tag to drive the framework content:

```
<cfinclude template="#request.tapi.process()#">
```

This sort of minimalist index template should be familiar to anyone using Fusebox, Mach-II, Model-Glue, or the Hub. What's different in this case is that we have many Web-accessible templates instead of having all page requests routed through index.cfm.

To be fair the same can be done also with any of these other frameworks, although the other frameworks are rarely implemented this way because they don't inherently gain any noticeable benefit from using this structure and as such it's generally easier to simply route all requests through the index.cfm template. However, with the onTap framework the name of the base template is used as a part of the execution path to determine which templates are executed by the framework. So using cart.cfm has an inherently different result than using index.cfm as the base template. Of course there's nothing preventing you from routing all requests through index.cfm, although using this multi-template strategy let me migrate the application more quickly thanks to not needing to edit any of the links, form targets, or cflocation tags in the application. Had I used any of these other frameworks, which use form/url parameters and XML configuration files to direct their execution, they would have required that I edit every URL in the application to add the appropriate event parameter (FuseAction, event, etc.).

Although I chose a directory structure that let me ignore URLs in the application, I chose not to take the same approach with images. Although I could have simply left the Pet Market images in the /ontap/petmarket/images/ directory and ignored any image tags in the application HTML, the framework provides some added features with regard to images and for this reason I decided to put the images in the framework's designated image directories.

Yup, multiple image directories.

Although most HTML applications put all images in a single directory, the onTap framework provides a method of easily targeting methods contextually to provide simple branding and replacement of images by context. Interestingly enough, the Pet Market application provides a perfect example of why I included this feature in the framework in spite of the fact that the original Pet Market authors never imagined that I would write this framework revision: more evidence that this is a good feature to have.

When you view the Pet Market application in a browser, you'll notice that the layout presents several images of different animals at the top, linked to category pages for the various categories of pets, such as dogs, cats, and reptiles. As you browse through the application, however, you might notice that a different set of images is presented on the home page than on any other page in the application. In the original application, the location of this image was tightly coupled with the layout custom tag: the layout tag contained logic to determine if the current page was the home page and displayed a different set of images (including the size of the image as part of the name of the graphic file). I put copies of each of these smaller five images in the directory /ontap/_components/petmarket/_images/category/ for most pages in the application to use, renaming them to remove the image size from its file name. This makes the image name succinct and canonical and the name of the image file accurate, even if the size of the image changes later. Then I copied the five larger images to the directory /ontap/_components/petmarket/index/_images/category/ for the home page to use with the same succinct file names.

Throughout the HTML for the application I then replaced all image tag SRC attributes with the function request.tapi.getIMG() containing the name of the target image. This function checks a series of nested image directories and returns the URL to the first image it finds. So when the layout requests request.tapi.getIMG("category/dogs.gif") it will use the image located in /petmarket/index/_images/category/ on the home page and on every other page it will use the image located in /petmarket/_images/category/.

You might notice the string "/index/" in the path of the images used in the home page, and you'd be correct if you guessed this string matched the name of the base template (index.cfm). In this way, I can loosely couple the branding of images for different pages without needing to add any logic to the display or layout templates to handle the variance. This way I eliminated all image logic from the layout template.

Then there's the issue of layout in general. As a developer I tend to think about every application in terms of portability. Call me stubborn; I refuse to make assumptions about an application's operating environment. This is one of the many weaknesses of the original Pet Market application, which required that you either add a custom tag path in the ColdFusion administrator or put tags in the default custom tags directory, neither of which are solutions I'm willing to accept for an application I've written.

In the case of layout, the application uses a custom tag to create the bulk of the HTML on each page. The custom tag is used with an end-tag to wrap the content of each page. For my application, this custom tag wouldn't be necessary, although much of the same structure can be reused. I copied the /extensions/customtags/wrapper.cfm template from the original application to /_components/petmarket/_layout.cfm and changed the logic for the tag internally to use the variable variables.tap.layout (which has a value of "header" or "footer") instead of using the ColdFusion native variable thistag.executionmode. Once this was done I removed the custom tag references from all the other templates in the /_components/petmarket/ directory. I also removed the head, body. and HTML tags from the layout template, since these are generated by the framework, and moved the content of the embedded CSS style tags to the template /_components/petmarket/_htmlhead/100_petmarket.css (which is included by the framework automatically in the resultant HTML).

In all honesty this isn't the way I'd normally handle layout. There's nothing wrong with it per se, although the onTap framework includes an HTML library – a set of functions for generating modular displays in HTML format with much greater flexibility – and normally I'd use this library via an XHTML custom-tag to create and cache the layout and then insert the content for each page into the generated display structures. Although this technique is more extensible, I simply didn't have time to implement it for this article (and Simon requested that I use the original application's view layer).

The original Pet Market application also puts its ColdFusion Components (CFC) in a directory not below the web root, requiring the creation of a ColdFusion mapping to target these CFCs. The onTap framework has a central repository for CFCs and a custom function for targeting these components that allows the application to be moved to any environment and configured quickly and easily (no need to change the extends attributes of the various CFCs).

For this I copied the original application's /extensions/components/petmarket/ directory to /ontap/_components/_cfc/petmarket/ then located all references to the ColdFusion native CreateObject() function and replaced them with request.tapi.getObject() (the arguments did not change). I also would have changed any references to the cfobject tag or cfinvoke tags that might use cfc paths instead of instantiated objects (and replaced their paths with request.tapi.getCFC()), although these aren't used in the original Pet Market application, making such changes unnecessary.

Finally, the Pet Market application included an Application.cfm template in its root directory (above the web root) to execute code at the beginning of each request. Although the onTap framework has its own Application.cfc for handling the various application events (onApplicationStart, onApplicationEnd, onSessionStart, onSessionEnd, etc.), the structure of this template lets us easily migrate code from other applications used in their Application.cfc or Application.cfm templates by copying the code into an appropriate subdirectory. In the case of my Pet Market migration, I copied the contents of the original Pet Market Application.cfm template to the file /ontap/_components/petmarket/_application/100_petmarket.cfm. This template will execute at the beginning of each request when the base template is in the /ontap/petmarket/ directory.

By the time this template executes, however, the framework has already instantiated the application and so the cfapplication tag is no longer necessary and I deleted it. If I wanted to change the name of the application or its timeouts or session management settings, I'd make those changes by setting variables in the structure request.tap.cf.app (i.e., request.tap.cf.app.sessionmanagement = true) using code in the template /ontap/_components/_appsettings.cfm. (This template doesn't exist in the core components archive to safeguard against overwriting your project settings when upgrading to the latest framework core version.)

At this point, I could have left the files as they were and the application should work. I prefer, however, to have the /_components/ templates a little further segregated, so I renamed each of the templates in the /_components/petmarket/ directory to _process.cfm and put them in another subdirectory with the name of the original template, i.e., /cart.

<SEEFUSION>™

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

cfm is renamed to /cart/_process.cfm.

Now I have my desired file structure:

```
ontap/
   _components/
      _cfc/
         petmarket/
   petmarket/
         _application/
         _htmlhead/
         _images/
         _layout.cfm
         about/_process.cfm
         cart/_process.cfm
         index/
             _images/
             _process.cfm
          etc/_process.cfm
   petmarket/
     about.cfm
     cart.cfm
     index.cfm
     etc.cfm
```

Object-Relational Mapping (ORM) and Data Access Objects (DAO)

The next task I tackled after moving and renaming all the original Pet Market files was to reinvent the manner in which the application accesses data. To be fair this wasn't entirely necessary. I could simply have left all the ad hoc SQL queries in place and claimed the application complete, although since I'm something of a perfectionist I wouldn't be happy with the application's existing structure for data access. The authors created a request variable (request.petmarketdb) for targeting the database with cfquery tags, which is a good start and where most applications end their encapsulation of datasources (not to be confused with encapsulation of SQL queries), although I'm personally unsatisfied with this minimalist approach. Instead, I created a Datasource Name (DSN) structure in the onTap framework for holding information about multiple databases, where I registered my Pet Market DSN (request.tap.dsn.petmarket).

I registered this structure in the request scope using the /ontap/_components/_appsettings.cfm template for general application configuration and then set about the task of replacing all the application's ad-hoc SQL queries with the onTap framework SQL-abstraction tools. By default these tools use the "primary" DSN (request.tap.dsn.primary) so if I want my SQL-abstraction code to access my new Pet Market DSN I have to include the DSN attribute or argument in each of the custom tags or function calls used to access this abstraction layer.

One way to reduce the amount of code needed to accomplish this task is to use a datasource.cfc provided by the framework and instantiate it with the name of the DSN I want it to use. Because I don't want to recreate this CFC on each request where I might need it, I create it in the application scope (application.petmerket.datasource) when the application starts by putting the object instantiation in the template /ontap/_components/_appstart/100_petmarket.cfm.

Once I started looking more deeply into the data access used in the Pet Market application, I realized that most of it could be replaced with standardized Object Relational Mapping (ORM) techniques. For this I used the onTap framework's dynamic inheritance tool in the soft constructor for most of the existing CFCs to extend the framework's Data Access Object (DAO) component (request.tapi.cfcExtend(this,"dao")) and eliminated a large amount of code from the CFCs that simply put data returned from ad hoc SQL queries into the "this" scope or a structure in the "this" scope.

Thanks to all of the code I could eliminate from these components, I reduced the CF/HTML/CSS code in the entire application from 85KB to 65KB. Although 20KB may not sound like a lot of code, it's roughly 25% of the entire application even after having added some of my own code (including a product-item DAO that wasn't included in the original application). That's a lot of keystrokes and time saved over a larger project.

The original Pet Market application included five CFCs: a _base.cfc that I eliminated (ontap.cfc provides an improved implementation of its only functional method), a shoppingcart.cfc for maintaining the current user's selected items and quantities and three components that are extended DAOs (category.cfc, product.cfc, and user.cfc). The original application stored product data in two tables (product and item), which prompted me to add the item.cfc for modeling data in the second table. The item.cfc also provided me with a good opportunity to highlight one of the advantages of the framework's dynamic accessors (getValue(propertyname)). In this case I could simplify the relationship between the product object and the product-item object by setting a property of "productname" in the item CFC's soft-constructor that ensures that the property is included in the structure returned from the getProperties() method and implemented the private get_productname method that instantiates the related product object and returns the value of its "name" property. So the item.cfc has a "productname" property that's drawn directly from the relevant related product object with no need for the accessing template to concern itself with the inner workings of these components or shape of the database schema.

Although the dynamic accessor feature let me eliminate many lines of code from the original CFCs, this altered structure forced me to check all of the display code to ensure that it accessed these objects using the appropriate methods (instead of arbitrarily exposed variables). For many of the display templates I opted not to replace the exposed variable references with dynamic accessor functions, chosing instead to append the desired variables to the attributes scope using the getProperties() method of the ontap.cfc and the ColdFusion native StructAppend() function. This strategy for decoupling variable use from both components and from the form and url scopes should be familiar to anyone who's used Fusebox in the past (and indeed I attempted to replace all form and url-scope variables in the application).

Show Me the Memory!

The original Pet Market application used many CFCs for different purposes, but it gained very little (if any) benefit from using CFCs. This is because the application created

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

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.

Jeremy Geelan

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

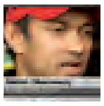
HURRY! LIMITED SEATING THIS SEMINAR WILL SELL-OUT CALL 201-802-3022 TO REGISTER!

Jesse James Garrett
Father of AJAX

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

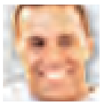
Bill Scott
AJAX Evangelist of Yahoo!

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 Hinchcliffe
Co-founder & CTO, Sphere of Influence Inc.


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

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

MEDIA SPONSOR
ajax developer's journal

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

PRODUCED BY SYS-CON EVENTS

these objects for each request and the components didn't encapsulate their data. So no advantage could be gained from caching the data for categories or products even though this data is rarely (or never) changed; in fact, the application didn't even include any tools for updating product data.
Since the onTap framework DAO objects are designed to make good use of caching techniques, I also implemented a factory object by extending the framework's factory.cfc. I instantiated the new petfactory.cfc and stored it in the variable application.petmarket.petfactory in the previously created template /ontap/_components/_appstart/100_petmarket.cfm that is executed when the application starts.

With this object in place I replaced all other instances of request.tapi.getObject() (previously CreateObject()) with calls to the methods of the application.petmarket.petfactory object (with the exception of user and shopping-cart objects instantiated at the beginning of a session). The petfactory object is now the only object that instantiates DAO components. As a result of this change, fewer queries are executed by the application, being limited only to search queries and queries executed when first instantiating a new DAO object. This caching makes the application significantly more scalable in handling large numbers of users and pet purchases. Since I created a factory object for the Pet Market application, I also took this opportunity to move the category-list and product-search queries to this factory object out of the category.cfc and product.cfc objects where they didn't belong in the first place.

A La Cart

I personally found Pet Market's implementation of a shopping-cart unacceptable for a number of reasons, both related to the model (data access and session management) and to the view (the checkout form is a horrid mess). I used several strategies to resolve what I saw as problems with the shopping cart.

First, I consolidated all of the code for the various checkout steps into the stepX.cfm templates in the /_components/petmarket/checkout/ directory so that each of these files contains all of the code for the relevant step of the checkout. The original version included only the form code in these files, while leaving code to display during subsequent steps casually misplaced in the checkout.cfm template. Then I removed the form-post code from the top of the checkout.cfm template and put it in several subdirectories of the /_components/petmarket/checkout/ directory. To target the code in these new directories I added a hidden form variable with the name "netaction" to each of the stepX.cfm form templates, indicating the relevant subdirectory. Now when each form is submitted only the relevant code for the submitted form is executed. Then I cleaned up /_components/petmarket/checkout/_process.cfm by putting the names of the steps in an array and used a loop and variables to generate the display of the list of steps and links to previous steps in the checkout process.

I must admit that the Pet Market shopping cart is an improvement over most shopping cart code I've seen because it smartly uses a query to hold the user's selected items (instead of the inappropriate multi-dimensional array used by most cart applications, which forces the programmer to use numeric indices where named keys would be more appropriate). Still, I think the shopping cart can definitely be improved.

Firstly, the display template /_shoppingcart.cfm fetches a query containing the names of all products in the database, which is then joined with the query containing the user's selected items. Although the first query is cached daily using the ColdFusion server's native query caching features, I find this technique to be unencapsulated and sloppy (the display code contains references to database schema) and I personally dislike the server's native query caching features. To alleviate these problems I implemented another of the framework's core CFCs – LinkedList.

To be honest I created the LinkedList.cfc as part of the framework core components specifically for handling memory-resident lists like shopping-carts. Although extensive, the transition from a ColdFusion query to a series of linked-list objects was reasonably simple thanks in part to the fact that several more complicated query tasks are encapsulated by the LinkedList.cfc object. When the cart is updated with quantities for one or more objects, checking to see if an item already exists in the cart is a simple function call (item.search("itemoid",arguments.itemoid)) instead of a query of queries. When an item is removed from the cart, a similar method (item.remove()) replaces another query of queries. Displaying the contents of the cart is a simple process of looping over the pre-populated items and displaying their names, quantities, and costs (while item.hasNext()), cleanly encapsulating the data access away from the display logic. And since the objects are pre-populated from the petfactory product objects prior to being cached in the session scope, this has the added benefit of ensuring that the names of products in the user's cart won't change before checkout is complete. Although the user.cfc still has a placeOrder method, the bulk of its work is now done by the appropriate shoppingcart.cfc (user.cfc is not an appropriate place for the bulk of this code).

The coup de grâce is a bit of incidental internationalization. That is to say, i18n implemented with no expenditure of effort. In the original Pet Market application, items in the cart display in the order in which they were added to the cart. I didn't like this, preferring that items in the cart be sorted alphabetically, so I implemented the LinkedList sort method (item.sort("productname")) each time an item is added to the cart (removing an item doesn't change the sort order and as such sorting is unnecessary). It merely happens that the sort method provided by the LinkedList.cfc uses Java international collations to provide non-lexigraphical sorting. That is to say, more than a simply case-insensitive sort, but rather the names of products are sorted using the dictionary and accenting rules of the user's preferred language.

About the Author

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

info@turnkey.to

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.

COMMUNITY MX

Intermedia.NET...

Now serving the hottest ColdFusion.

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
- Competitive plans
- Security sandboxes
- Custom tag registration
- Verity collections search engine
- 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