

EVEN WEB SPIDERS ARE AFRAID OF...

U.S. \$8.99 (Canada \$9.99)

COLD FUSION Developer's Journal

ColdFusionJournal.com

January, 2000 Volume: 2 Issue: 1

Your Favorite
ColdFusion
Information
Resource Is Now
MONTHLY!

Editorial **Comments Anyone?**

Robert Diamond page 5

WDDX

CFWDDAP: Writing Custom WDDX Interfaces to LDAP

Robert Fielding &
Sarge Sargent page 20

Foundations

Improve Your Code Readability with Fusedoc

Hal Helms page 48

CFDJ NEWS
page 54

CFDJ Marketplace
page 52

RETAILERS PLEASE DISPLAY
UNTIL FEBRUARY 29, 2000



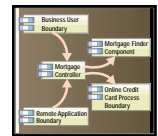
SYS.CON
PUBLICATIONS



CFDJ Feature: Architecting Enterprise Level ColdFusion Applications

***Inadequate design time can cause useful applications
to fail - what they need is a good road map***

Ed Donohue & Ben Elmore



6

<BF> on <CF>: And It Just Keeps Getting Better 16

***Don't the new half version number change fool
you. This is significant and impressive update***



Ben Forta

CFDJ Special Feature: ColdFusion with Classes 24

Construct complex enterprise systems that perform

Ralph Fiola

CFDJ Special Feature: Virtual Arachnophobia 32

Even web spiders are afraid of something



John Morgan

CFDJ Feature: An Online Ticket Store-- The Storefront

***Using ColdFusion and the Java
Platform to build a simple Internet-based ticket store application***



38


Ajit Sagar

AbleCommerce

The flexible solution for building  electronic shopping sites

AbleCommerce

CLIENT LIST

A few of our
 shopping customers

24 Hour Fitness
Bushnell
Johns Hopkins University
BOC Gases
Key Tronic
Casio
Citgo
GSA
Lionel Trains
Lucky Brand Jeans
Compaq
Mercury Marine
Dairy Farmers of America
Mitsubishi
FAA
Nike
Garth Brooks
Miss Universe/USA/Teen USA
NorWest Financial
George Strait
PGE
Edwin Watts Golf Shops
Quest
Hasbro
Siemens
U.S. Department of Treasury
International Paper
United Nations
Jimmy Buffett
Jeff Gordon
Verio

To see more, visit
www.topsites.com

Complete Site Building and Administration from your browser.

Build "your_store.com" with secure Merchant Credit Card Processing. Maintain inventory, add discounts and specials to keep your customers coming back. Increase sales with cross selling and membership pricing. For accurate shipping use AbleShipper™ UPS auto-calculator with the flexible AbleCommerce™ shipping tables. Fulfill document and software purchases through electronic delivery. Check orders online, generate automated email and fax notification.



Enterprise Scalability for stores of any size.

Supports Microsoft Access and SQL. Add store and server license packs to create an unlimited number of stores. Two security levels for system and merchant administration allow ISP's to empower renters with online store design and maintenance abilities.

You can also integrate and customize AbleCommerce™ with open source code.



Easy Design Interface for eye catching sites.

Upload graphics directly to your server. Use prebuilt store templates for quick design. Customize your web pages and styles dynamically. Add you own HTML, Java, Flash or any other script to create a unique look. Add comment forms and surveys with automated responses to collect data from your visitors. Track visitor click throughs to determine the strengths of your site design.



"AbleCommerce is a perfect example of ColdFusion's strength and flexibility."

-Ben Forta, Allaire Corp.

"The system is easy to use and very flexible from both a customer and administrator standpoint."

-Ron Derby, Casio, Inc.

TRY IT OUT FREE!

Download the 30-Day Evaluation

www.ablecommerce.com

AbleCommerce, 11700 NE 95th Street, Suite 100, Vancouver, Washington, 98682, (360) 253-4142

AbleCommerce is a trademark of Able Solutions Corporation. ColdFusion is a trademark of Allaire Corp. Company or product names referenced may be the trademark or registered trademark of their respective companies. © Able Solutions Corporation, All Rights Reserved.

computerjobs.com

www.computerjobs.com

conceptware ag

www.conceptware.com

STEVEN D. DRUCKER, JIM ESTEN, BEN FORTA,
STEVE NELSON, RICHARD SCHULZE, PAUL UNDERWOOD

EDITOR-IN-CHIEF ROBERT DIAMOND
ART DIRECTOR JIM MORGAN
EXECUTIVE EDITOR M'LOU PINKHAM
PRODUCTION EDITOR CHERYL VAN SISE
ASSOCIATE EDITOR NANCY VALENTINE
PRODUCT REVIEW EDITOR TOM TAUILLI
TIPS & TECHNIQUES EDITOR MATT NEWBERRY

WRITERS IN THIS ISSUE

ROBERT DIAMOND, ED DONOHUE, BENJAMIN ELMORE,
BEN FORTA, ROBERT FIELDING, RALPH FIOL, HAL HELMS,
JOHN MORGAN, AJIT SAGAR, SARGE SARGENT

SUBSCRIPTIONS

SUBSCRIBE@SYS-CON.COM

FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS,
PLEASE SEND YOUR LETTERS TO
SUBSCRIPTION DEPARTMENT.

SUBSCRIPTION HOTLINE 800 513-7111

COVER PRICE \$8.99/ISSUE

DOMESTIC \$79/YR. (12 ISSUES)

CANADA/MEXICO \$99/YR.

OVERSEAS \$129/YR

BACK ISSUES \$12 EACH

PUBLISHER, PRESIDENT AND CEO FUAT A. KIRCAALI
VICE PRESIDENT, PRODUCTION JIM MORGAN
VICE PRESIDENT, MARKETING CARMEN GONZALEZ
ACCOUNTING MANAGER ELI HOROWITZ
ADVERTISING ACCOUNT MANAGER ROBYN FORMA
ADVERTISING ASSISTANT MEGAN RING
ADVERTISING ASSISTANT CHRISTINE RUSSELL
GRAPHIC DESIGNER ALEX BOTERO
GRAPHIC DESIGN INTERN AARATHI VENKATARAMAN
WEBMASTER BRUNO Y. DECAUDIN
WEB EDITOR BARD DEMA
WEB SERVICES INTERN DIGANT B. DAVE
CUSTOMER SERVICE ANN MARIE MILILLO
JDJ STORE.COM JACLYN REDMOND
ONLINE CUSTOMER SERVICE AMANDA MOSKOWITZ

EDITORIAL OFFICES

SYS-CON PUBLICATIONS, INC. 39 E. CENTRAL AVE.,
PEARL RIVER, NY 10965
TELEPHONE: 914 735-7300 FAX: 914 735-6547

COLD FUSION DEVELOPER'S JOURNAL (ISSN #1523-9101)
IS PUBLISHED MONTHLY (12 TIMES A YEAR)
FOR \$49.99 BY SYS-CON PUBLICATIONS, INC.,
39 E. CENTRAL AVE., PEARL RIVER, NY 10965-2306.

POSTMASTER

SEND ADDRESS CHANGES TO:
COLD FUSION DEVELOPER'S JOURNAL
SYS-CON PUBLICATIONS, INC.

39 E. CENTRAL AVE., PEARL RIVER, NY 10965-2306

© COPYRIGHT

COPYRIGHT © 2000 BY SYS-CON PUBLICATIONS, INC.

ALL RIGHTS RESERVED. NO PART OF THIS PUBLICATION MAY BE
REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS,
ELECTRONIC OR MECHANICAL, INCLUDING PHOTOCOPY OR ANY
INFORMATION STORAGE AND RETRIEVAL SYSTEM,
WITHOUT WRITTEN PERMISSION.

FOR PROMOTIONAL REPRINTS, CONTACT REPRINT COORDINATOR.

SYS-CON PUBLICATIONS, INC., RESERVES THE RIGHT TO REVISE,
REUBLISH AND AUTHORIZE ITS READERS TO USE
THE ARTICLES SUBMITTED FOR PUBLICATION.

WORLDWIDE DISTRIBUTION

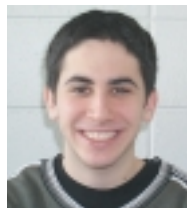
BY CURTIS CIRCULATION COMPANY 739 RIVER ROAD,
NEW MILFORD NJ 07646-3048 PHONE: 201 634-7400

DISTRIBUTED IN USA

BY INTERNATIONAL PERIODICAL DISTRIBUTORS
674 VIA DE LA VALLE, SUITE 204, SOLANA BEACH, CA 92075
619 481-5928

ALL BRAND AND PRODUCT NAMES USED ON THESE PAGES
ARE TRADE NAMES, SERVICE MARKS OR TRADEMARKS
OF THEIR RESPECTIVE COMPANIES.

Comments Anyone?



BY ROBERT DIAMOND


Somewhere in my life, apart from being editor-in-chief of **CFDJ** and running the Web Services Department at **SYS-CON Publications**, I have a bit of spare time during which I work on a few other Web sites. One of them is the official site for entertainer Michael Crawford (the original Phantom of the Opera). Recently I was called on to develop a new, secure Web board, online directory, and conferencing and messaging system for members of the site to use, along with a secure, unique login for each of them.

My first thought was to use Allaire Forums, a great product that runs great on **SYS-CON's** Web site. However, it couldn't provide me with exactly what we were looking for, so I decided to take on the task of creating one myself. I started off with lots of handwritten notes on the basic logic schema of how well it would work, beginning with the registration, then processing it to allow access to the board and e-mail notifications and then to the actual workings of the board itself. I also mapped out what the structure of the database would be, including tables, queries and so on. Then it was time to start coding. After a few weeks – and after a few late nights of cursing and trial and error (something I'm sure almost everyone can identify with) – I had a basic functioning system. Next came the bug-squashing phase in which I enlisted the help of everyone I knew and even a few people I didn't till finally everything worked as it was supposed to. Then came the final phase of making it user-friendly, converting it from something designed for and by a programmer to something your average person could use.

I submitted the URL and instructions to the powers that be, who made a few minor changes – “little” changes from their point of view, but major changes in development from mine. After I finished whining and making the adjustments, I was done – the project was complete.

The system went online, and worked amazingly well. A few weeks later I got a call from a friend who was trying to set up a similar system for a project he was working on. I was very busy at the time, so I offered to send him my code, an offer he happily accepted. I sent it over to him and didn't hear back from him for about a week. When I did, the message was short: “Thanks for the code. I accept the fact that it works, I really do, but you put no comments in and I have no idea how it works. I can't use it.” (There were a few expletives in there too, but those have been removed.) I dug out my pile of original notes and faxed them over to him, thinking that seeing my logic would help. It didn't. I'd only managed to confuse him even more. Finally he gave up and – after much more swearing at me – did it all himself.

I guess the moral of my little story is that no matter how great a coding job you do, without comments the code isn't of much use. To add weight to this statement, while working on this editorial I looked back at the code I'd written: large chunks of it are Greek to me too. Some of the “roundabout” logic that seemed so brilliant to me a few months ago still has me scratching my head. I am happy to report that I commented my next two projects.

I'm *unhappy* to report that since then I've fallen back into my old, bad habits. Time is money, folks, and the idea of cutting a corner to make something easier was just too appealing. Have a similar experience? Drop me a line at rdiamond@sys-con.com or post it on our forums at www.ColdFusionJournal.com. Happy coding! 

ABOUT THE
AUTHOR
*Robert Diamond is
editor-in-chief of
ColdFusion Developer's
Journal.*

Architecting Enterprise Level ColdFusion Applications

Part 2

Inadequate design time
can cause useful apps to
fail – what they need
is a good road map

BY ED DONOHUE & BEN ELMORE

Ultimate Components

In our previous article in *ColdFusion Developer's Journal* (Vol. 1, issue 5) we discussed the importance of thinking in terms of components. This article expands on the idea of component-based architectures by showing how to create the road map you'll need when building your applications.

We'll look at and describe the overall development life cycle of our Web applications and focus on the steps involved with crafting this architecture in order to illustrate object modeling while mapping it to both ColdFusion and Spectra applications. Through the use of a component-based architecture, we'll save time and improve the overall quality of our software.

Design Is Paramount

Why do applications that start out on the right track fail? Why do applications break the first time they're enhanced? Why do terrific applications need to be rewritten every time business requirements change?

The common answer to these questions is simple: limited design time was given to these applications. Useful applications are constructed all the time but quickly fail due to the inability to read into the future and foresee change. They lack an adequate road map. However, good

design allows you to react to changing business requirements, facilitate enhancements and integrate new technologies in a timely manner without impacting the current systems. The need for architecture and design can be summed up by the following examples.

Imagine a typical child's bedroom closet filled with toys, balls, bat, etc., piled one on top of the other. Unfortunately, this is the way some applications (both Web and client/server) are developed. If a small boy went in and removed an item, he *might* be able to shut the door quickly before anything fell out, but the next time 'round....

Now picture the same closet after we put a little architecture and design against it. The same closet has built-in compartments, with each toy in its own compartment.

Which closet would you prefer your application to look like?

Identifying Our Road Map

Like any good building, applications need to begin with a solid set of blueprints built on firm foundations and implemented through a planned-development life cycle. The software industry recently adopted an iterative approach to application development. With this approach we're able to structure our applications to facilitate minideployments that allow us to get to market faster. Figure 1 shows the life cycle of our applications as it relates to a series of sections, phases and steps. In this way our approach is structured as Sections that contain defined Phases made up of sequential Steps.

The development life cycle is broken up into three major sections: *Predevelopment*, *Development* and *Postdevelopment*.

The primary purpose of the Predevelopment section is to define the scope of the application. The next section is the actual application development cycle – a combination of incremental design, implementation, testing and deployment steps. This section is looped over for each case (for the system usage) that yields a set of minideployments culminating

in a final production-ready system. The last section is where we'll do the monitoring and performance of our sites. Throughout these sections you'll notice recurrences of the design, construction and deployment phases.

In the design phase we'll draw up the blueprints for our applications, including the first nine steps depicted in Figure 1 (we'll expand on these steps later in the article).

Steps 10-13 represent the construction phase in which the foundation and business portions of our applications are built.

In the final phase, Steps 14-17, the application is promoted from the development environment to testing and then finally into production.

The important point to note here is that our applications are constantly moving between or spanning these three phases in our development life cycle. As managers we can focus the efforts of our developers into any part of the development cycle at any time. It's quite possible to concurrently have one team flushing out use cases into sequence diagrams, another team working on constructing objects from the class diagram, and a separate team working on deploying a version of the application. In Spectra, applications are constructed through the use of participants. Thus we're able to map and track where each of the participants are and manage where they come into play.

The remainder of this article focuses on the design steps involved in our application development and the best way to understand them, through an example. Let's say Company X wants to build a Web-based Mortgage Rate Finder application and has come to us for help in designing the application.

Step by Step

Steps 1, 2, 3 and 4 of the design phase (see Figure 1) need to be completed in sequential order and heavily involve both you and the client. Plan for plenty of time for these steps and remember: time spent up front shaves time off the end.

We first need to gather everything necessary for the application. This involves sitting down with the client and really finding out what he or she *wants*, *needs* and *expects* the application to do. This will allow us to identify the boundaries of the application, prevent scope creep and handle misconceptions up front.

The requirements for the Mortgage Finder application are (please note that we kept the requirements minimal):

1. To find mortgages through a Web interface
2. To be able to filter the types of mortgages desired through properties
3. To be able to use the mortgage finder through a remote application (Application Syndication) and return the results as data (data syndication)
4. To charge the users for use of the system

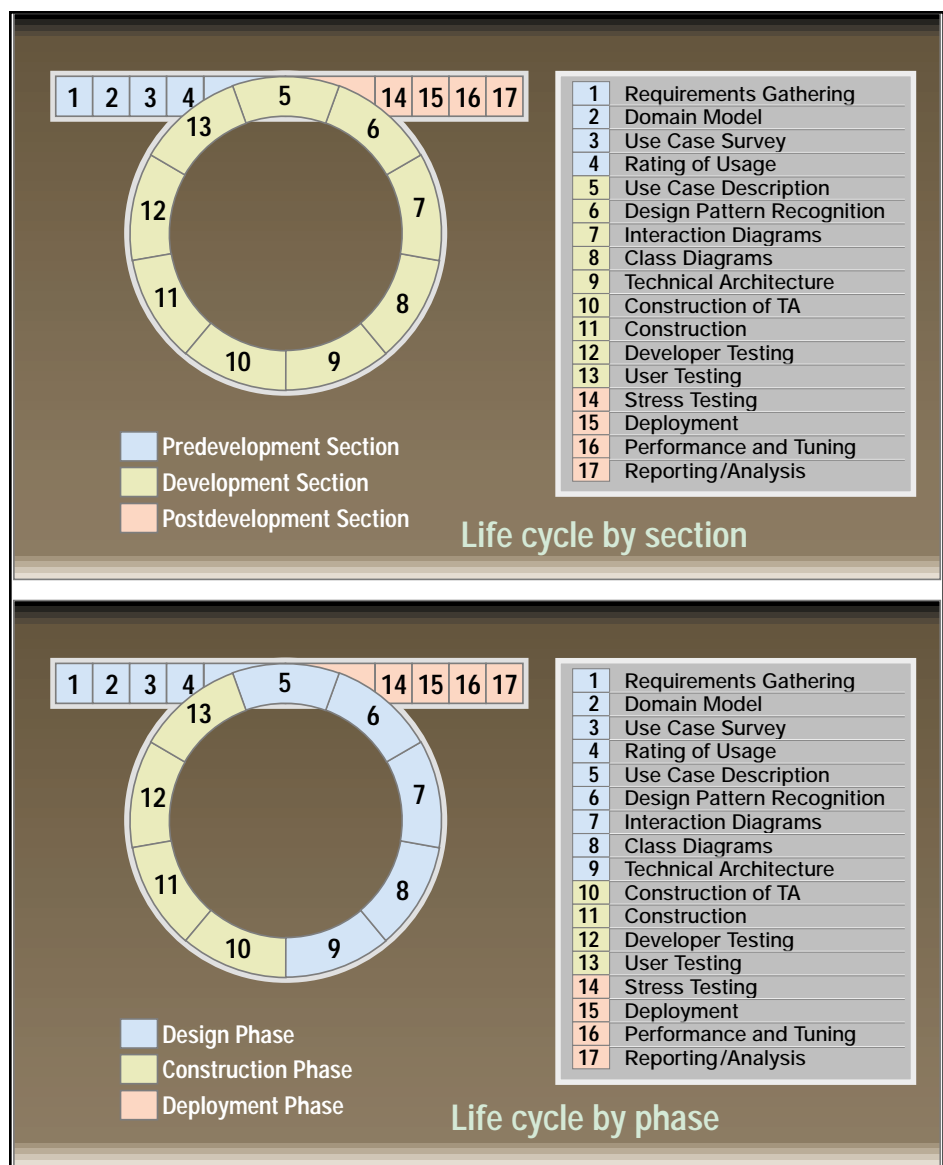


FIGURE 1: Development life cycle

Once the first step has been taken, we'll be able to come up with a base domain model that'll visually show us the boundaries, processes and responsibilities of our system. At this point we suggest you keep elaboration to a minimum (this will be done later in the process). Figure 2 shows the domain model for the example application.

The domain model begins by looking for boundary points (components). Boundary points depict where the user interacts with your system, where the system interfaces with outside systems and where the new systems interact with existing applications. In our example the boundary points are the *online credit card processing*, the *business user* and the *remote application*.

The next items we look for are business areas such as product, orders or employees. All the transactional logic and rules of the business areas are located here. In our example application the business area is the mortgage finder component. When identifying business

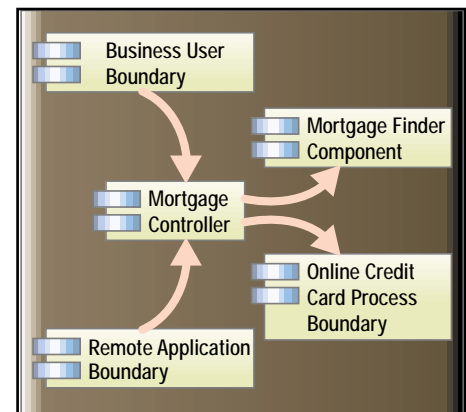


FIGURE 2: Domain model

areas, be careful not to join two together that are truly separate just because they feel similar. Last, you need to map out controller components that'll coordinate requests between the boundaries and the multiple business areas. This is the beginning of the concept of application partitioning. The logic behind this

step is to keep it simple and realize that this model might change as we flesh out the detail later in the process.

- It's easy to think of ways to bypass this model such as directly accessing the ColdFusion templates.
- Recognize there are issues with the language implementation.
- The architecture we're creating should transcend languages and implementation (Spectra or ColdFusion).

Now we can create an alternate view of our application to tentatively show what the site will look like. This view, which shows the user interface of the application, can also be referred to as a UI prototype. Unlike the component domain model, this effort isn't meant to focus on the process of the business. Instead, it allows us to present to the user something tangible in a short amount of time. We recommend that you spend very little time creating this view. There are many tools on the market that can help you create the basic site layout. Figure 3 shows the site layout for the example application. It's also important to make sure clients don't get hung up on this stage since this is merely a prototype to be modified as needed.

Different participants want and need different views of the same model. Therefore, the importance of each model's view is dependent on who's viewing it.

Next, we take the requirements and the Domain Model to formulate a Use Case Survey (also known as a Use Case Diagram). Here's where we identify the possible uses and actors of our system. Unlike the Domain Model in which we identified components, the Use Case Survey identifies interactions with the system. Figure 4 shows the Use Case Survey for the example application.

An actor, represented as a star, is defined as "a role that a user plays with respect to the system" and includes both people and other applications. Using the Domain Model's boundary components, we can quickly identify the actors in our system. On further examination, we might need to split the boundary component – which interfaces to the user – into multiple actors. For example, the application that interfaces to employees could have certain interactions that are available to commissioned employees only. In that case you would get two actors: *commissioned* and *salaried*. In the example application we have three actors: Site Member, Site Affiliate (Remote Application) and the Online Credit Card Processing Application.

A use case, represented by the oval, is defined as "a typical interaction between a user and a computer system." The primary identifier of a use case is that it achieves a goal for the

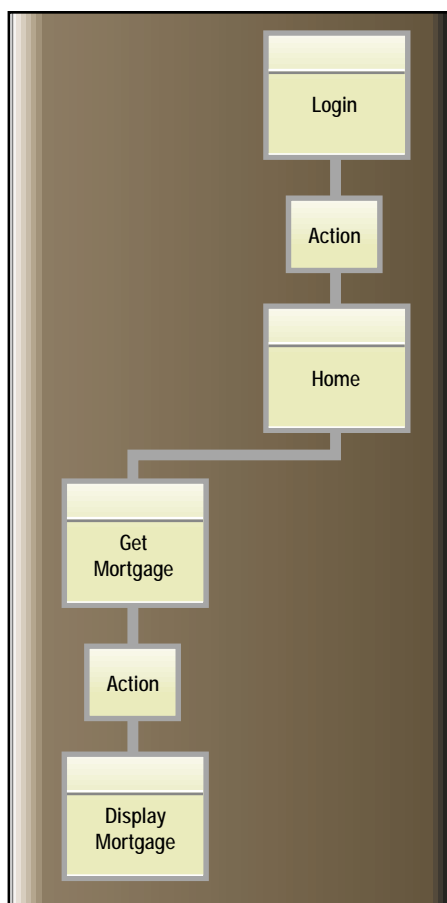


FIGURE 3: Site layout model

actor. You may find that there's no one-to-one mapping of requirement to use case, since use case will tend to encompass more than one requirement. An example of this situation would be, Do "Add employee record" and "Update employee record" belong in the same use case? We would say yes. The objective of the user is to maintain the employee records. The goal of identifying use cases is to keep them at the highest level possible. From a Spectra standpoint these will correspond to some of your workflows and Process Logic Paths (PLPs).

The fourth and final step before diving into iterative development is the prioritizing of the use cases identified in the survey. At this point with the client, each use case is ranked according to its technical and business perspective. For instance, the most technical and business-critical use cases should be ranked so that they're completed first, to minimize the risk of not completing the core parts of the application on time. To help with the prioritizing effort, it would be prudent to have a solid idea of the deployment strategy ahead of time. Identify how much time is needed to complete the whole project, and when the application needs to "go to market." This will determine the number of versions you'll have to prepare for. For example, if the length of the project is eight months and the client needs functionality releases every two months, we'll need four versions. Each will include the number of use

cases assigned during the prioritizing. When evaluating use cases, see whether they require immediate attention. If not, schedule it for a later version. Skipping this step and moving into the construction phase cripples our efforts to have incremental releases and could potentially leave our applications open to risk. This will also marshal development efforts into areas that won't be released until later. The ordering of the use cases for our example application is shown in Table 1.

We can now begin the development section of the development life cycle. It's possible to complete a step for all the use cases before moving to the next step, but you'll get better results and bring the application to market faster if you develop iteratively. Processing a single use case through each step does this. *Note:* It's important to continue to develop specifications with the client's help. All steps should be completed and approved by the client. Using Spectra we can build the approval process right into the Webtop using workflows and PLPs. Pretty cool...an application that manages its own completion!

Step 5 is to take each use case and create a Use Case Description that describes it in more detail. All possible paths of execution are identified and expanded on. The three categories of paths are *Primary*, *Alternate* and *Error*. The Primary path (a.k.a. "happy path") depicts the default path employed when the user invokes the use case. The alternate paths are used to show additional actions or branches to the primary path. The error paths show the paths that execute only if an error occurs; they also identify unrecoverable errors. During this process of identifying the steps for each path,

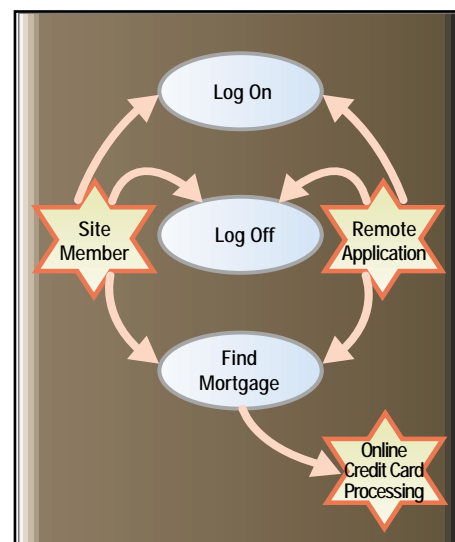


FIGURE 4: Use Case Survey

Use Case	Number	Version Release
1. Log On	RS001	Version 1
2. Log Off	RS002	Version 1
3. Find Mortgage	RS003	Version 1

TABLE 1: The ordering of the use cases

datareturn

www.datareturn.com

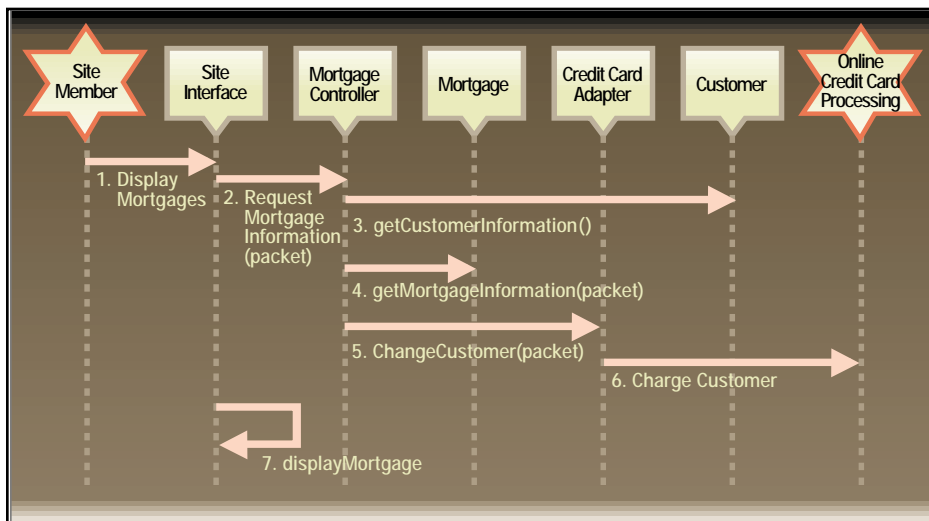


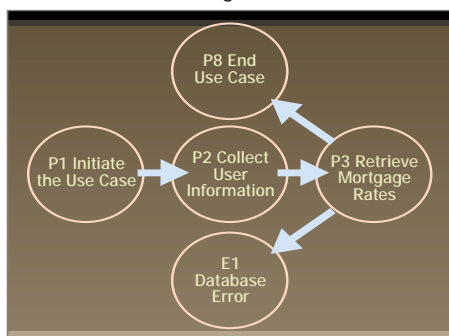
FIGURE 5: Interaction diagram for the “Find Mortgage” use case

we’re documenting each step and identifying what’s needed for it to complete successfully. By keeping everything at a high level, we can continue to involve our clients with the design process. This step in the design will help cut down on the number of change requests made later because of missing logic or unclear areas. The use case description will also provide the definitions to our Spectra workflows and PLPs. An abbreviated use case description for the “Find Mortgage” use case showing the Flow of Event (F.O.E.) and a documentation outline is shown below. Use case descriptions can be used to come up with more concrete user interface examples that the user interacts with.

1. Use Case Report: Find Mortgage – ID# RS003

This use case describes the functionality used to find the specified mortgage based on user input.

- 1.1 Preconditions
 - 1.1.1
- 1.2 Flow of Events
 - P1) Initiate Use Case: ...
- 1.3 Alternative Flow of Events
- 1.4 Exceptions and Resolution
 - E1) Database Error: ...
- 1.5 Special Requirements
- 1.6 Postconditions
- 1.7 Pictures of the User Interface
 - 1.7.1
- 1.8 Notes
- 1.9 Flow of Events Diagram



Step 6 is Design Pattern Recognition. The subject of design patterns will be explained in more detail in our next article. This step is simply a way to look at the use case description and determine if any of them contain a common problem(s). At this point we can bring in a set of templates that have been shown to solve the problem.

Step 7 is to create Interaction Diagrams. This takes each path of the use case description and maps it to the component responsible for the logic and what the action on that component is. Use the Domain Model to help identify the components and their responsibility. Here we identify what component triggers the logic on the next component, and what attributes need to pass between the components. The interaction diagram for the primary path of the “Find Mortgage” use case is in Figure 5. The boxes on the top denote a component, and the arrows between the components show the action fired on the component at the end of the arrow. Each action is named, and the needed information is enclosed in parentheses after the name. This is to show the action that each component needs to complete, not actually code.

Step 8 is to create a Class Diagram that depicts the components that’ll be involved in the use case. The list of components will be composed of both new and existing components depending on what iteration you’re on. As we build these components, we’re amassing a repository of building blocks to use and reuse, thus using the manufacturing model described in our earlier article. This will break down business areas, controller components and boundary components into the components that constitute them. At this time we need to make a decision about the application. Do we create a separate component to display a component’s information, or do we allow the component to display the result itself? From a Spectra standpoint this step will help identify the content Objects of the application and

whether we need to create a separate one for display purposes.

The last step in this phase, designing the technical architecture, will consume the most time during the first couple of iterations. This is where system services (personalization, error, caching) are designed and decisions concerning exception handling, memory management, clustering issues, page caching, data accessing and so on are made. This step introduces issues that are language dependent. For example, Spectra contains prebuilt services we can use, such as remote invoking, so we won’t need to design our own. Issues could include whether to use client or session variables, or <CFMODULE> instead of <CF> to access custom tags and development standards.

UML and CF

Class Diagrams are a way for us to design and view our applications before the first line of code has been written. In order to read the diagrams we need to understand some basic UML (Unified Modeling Language) object notation. We also want to stress that ColdFusion isn’t object-oriented; at best, ColdFusion can be made object-based. As with any other programming language, ColdFusion behaves the way you make it.

The base of a domain model is its classes (objects) and their relationship to other classes. In Figure 6 you’ll find the symbol for a class, which is broken up into three compartments that represent the name, properties and methods. For those of you working with Spectra, this makes up the Object Type Definition.

In straight ColdFusion development we don’t have access to all of the advantages of true object-orientation. Instead we must create a file-based way to duplicate it. Figure 7 shows the files that would be created for the above object. Creating an interface file that calls the appropriate method through a custom tag – depending on the method name – mirrors the behavior of an object.method call. Listing 1 shows the control structure for the interface object along with how it’s invoked.

Objects relate to each other in three basic ways: inheritance, association and aggregation. In *aggregation* the component owns a separate component and is in charge of “creating” it. The only type of aggregation we have in ColdFusion deals with creating objects using the <CFOBJECT> tag. *Inheritance* is the ability to derive new (more specialized) classes known as *subclasses* from an existing class known as a *superclass*. A subclass receives all of the superclass’s behavior and attributes. This provides us with the ability to create generic classes (such as a site personalization class) and a more specialized class that inherits all of the properties and methods. This saves us the pain of maintaining duplicate code that would otherwise need to be created in both components. In ColdFusion duplicating

eprise

www.eprise.com

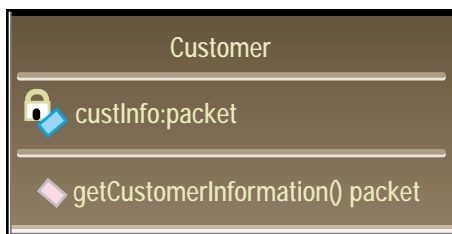


FIGURE 6: Class Object

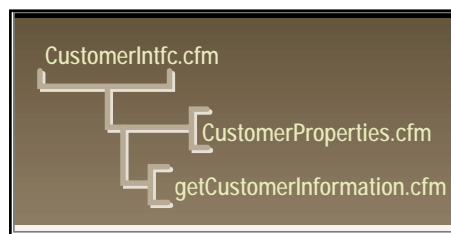


FIGURE 7: File structure for CF representation of object



FIGURE 8: The three forms of relationships

this functionality is a little trickier. It's accomplished by having the interface file call the same method on the interface file of the component it inherits from. See Listing 2 for the control structure to do this. Finally, *association* refers to a relationship in which one component knows about another component. This is the most common relationship found in our ColdFusion applications. Figure 8 shows the syntax representation of each of the three relationships.

Two more concepts we should discuss are polymorphism and encapsulation. *Encapsulation* is the grouping of like data and behavior together within a single class (component) with a single purpose. It hides its implementation details within it and provides a well-defined public interface to get at that class's (component's) capabilities. When change is needed, it's only in one place. *Polymorphism* is the implementation of similar behavior via the same message structure across different classes (components). Using the `<CFMODULE>` tag we can build the name of the component dynamically but leave the method name the same. See Listing 3 for an example. In Spectra this is done by direct invocation against an undetermined Object Type.

To apply this approach, it's imperative that we think of our application as components. At

that point it makes sense to encapsulate the appropriate component. It'll also be helpful to have the following understanding of ColdFusion structures: nested custom tags, subchild tags, `<CFMODULE>`, WDDX and variable scoping. As we continue with these articles, we'll touch briefly on these areas.

It Works

A couple of questions are raised at this point. First, How much of this can I really use? More than you think! Is this overkill? No, not if you're concerned about the future. This approach isn't meant to take our software development cycle and make it as long and tedious as the client/server would. Our plan is threefold: (1) we want to help current client/server programmers transition smoothly into Web design by giving them some common ground; (2) we hope to steal some of the hard-earned lessons from the client/server arena and use them in our application development; and (3) we're looking to steer you toward proper Spectra development. Think of these as guidelines for your development. We've had the pleasure of working with Spectra, and proper design will be a requirement in order to build successful applications with this product. Even putting Spectra aside, the benefits are enormous.

Looking Forward

In the next article we'll examine the use of design patterns and their roles in the development cycle. We'll take a look at how our design fits into the partitioning model, and how we can use the idea of application servers to reuse logic across enterprises. We'll also look at how to take this design and move forward.

For further information on these techniques, we recommend *UML Distilled: Applying the Standard Object Modeling Language* by Martin Fowler et al. (Addison-Wesley). Also note that we created the diagrams with Rational Rose and the site layout model with Visio.

Special thanks to Robert Crooks at Allaire and Kevin Webb at CGI Information Systems for their time and insights.

About the Authors

Ed Donohue, a certified Allaire instructor, is a national speaker and Internet evangelist with over 16 years' experience designing and developing applications across a wide variety of industries.

Benjamin Elmore is the founder and CTO of RS Technologies. Ben is also a certified Allaire instructor and a certified Powersoft developer associate.

e@edonohue.com ben@remotesite.com

LISTING 1: Control structure for the interface component

```

<!--
FileName:
    customerIntfc.cfm

Purpose:
    Component Interface Page

Description:
    Route the method calls to the appropriate methods

Attributes: (Varname - Datatype, (Required/Optional),
    (Accepted Values), Description about use)
method - String, (Required), Determines method to fire
sAttributes - Structure, Optional, Attributes to pass to the
    appropriate method

Modification: (Date, Name, Description)
    Date, Name, Description

Return Values: (Varname - datatype, (Required/Optional),
    (Accepted Values), Description about use)
Varname - Datatype, (Required/Optional), (Accepted Values),
    Description about use

Exceptions: (Value, Reason, Severity)
  
```

```

1001 - Invalid argument passed, HIGH
1002 - Required argument missing, HIGH
2001 - Unknown Method requested, Medium
3000 - Other, HIGH

Created By:
Name: Benjamin Elmore
Date: 11/22/99

--->

<!-- Check attributes --->
<cfif Not isDefined("attributes.method")>
    <cfthrow message="Attribute 'Method' not found." error-
Code="1002">
</cfif>
</cfif>
<cfif Not isDefined("attributes.sAttributes")>
    <cfset attributes.sAttributes = StructNew()>
</cfif>
<cfif not isStruct(attributes.sAttributes)>
    <cfthrow message="Attribute 'sAttributes' is not a structure."
    errorCode="1001">
</cfif>
</cfif>
  
```


ektron

www.ektron.com

LISTING 2: Control structure for inheritance

```
</cfif>
</cfif>

<!-- Include the component properties --->
<cfinclude template="customerProperties.cfm">

<!-- Check the method attribute --->
<cfswitch expression="#attributes.method#">
<!-- Get customer information --->
<cfcase value="getCustomerInformation">
<cftry>
<!-- Call method and pass properties and attributes --->
<cfmodule template="getCustomerInformation.cfm"
    sAttributes="#attributes.sAttributes#"
    sCompProperties="#sCompProperties#"
<cfcatch type="Any">
<cfthrow message="#cfcatch.detail#" errorCode="3000">

</cfcatch>
</cftry>
</cfcase>

<!-- Unknown method requested, check on ancestor--->
<cfdefaultcase>
<!-- Set the found boolean to FALSE --->
<cfset lblFoundMethod = "1">

<!-- Fire method on Ancestor --->
<cftry>
<!-- Fire on Ancestor --->
<cfmodule template="baseIntFc.cfm"
sAttributes="#attributes.sAttributes#"
method="#attributes.method#"
<cfcatch type="Application">
<cfif cfcatch.errorCode is "2001">
<cfset lblFoundMethod = "0">
<cfelse>
<cfthrow message="Unknown error from parent object"
errorCode="3000">

</cfif>
</cfcatch>
<cfcatch type="Any">
<cfthrow message="Unknown error from parent object"
errorCode="3000">

</cfcatch>
</cftry>

<!-- Check to see if found method --->
<cfif not lblFoundMethod>
<cfthrow message="Unknown method requested." errorCode="2001">

</cfif>

</cfdefaultcase>

</cfswitch>
```

```
<!-- Create struct with variables -->
<cfset sLocAttributes = StructNew()>
<cfset sLocAttributes.custName = "Ben">
<cfset sLocAttributes.ccCardNum = "999999999999">
<cfset sLocAttributes.sShopCartItems = attributes.sShopCartItems>

<!-- Fire pay on current online transaction adapter -->
<cfmodule template="#sCompProperties.compCurrentOnlineTransactionAdapterReference#" method="pay" sAttributes="#sLocAttributes#">
```

The code listing for
this article can also be located at
www.ColdFusionJournal.com

funderere software

www.funderere.com

And It Justs Keeps Getting Better

BY
BEN
FORTA



ColdFusion 4.5 is now publicly available. But don't let the half-version number change fool you. This new version is one of the most significant and impressive updates thus far.

I'm not talking about the obvious enhancements, things like:

- *Linux version*
- *Native Solaris code*
- *Service-level failover*
- *Cisco LocalDirector integration*
- *Server-side Java support*

Sure, those are all important and exciting. But amid all that excitement I fear some of the other really cool enhancements (ones that are immediately usable, ones that offer immediate value and benefit) are being overlooked.

So, without downplaying the above-mentioned features, here is my own list of reasons to upgrade to CF4.5 immediately.

Improved Variable Locking

This is an important one. ColdFusion features several forms of persistent variables (see Table 1) – that is, variables that remain and retain their contents from one client request to the next. Persistent variables are important. They're about the only way to keep data (user information, shopping cart items, etc.) as users work their way through your site.

But the use of persistent variables comes with a risk. Because ColdFusion is built on a multithreaded, multitasking engine, there's a real chance of mul-

tipple concurrent access attempts to the same data. If all you ever do is read this data, you'll probably never run into problems, but write access is a whole different ball game, and one that's likely to result in data corruption.

To help address this problem, ColdFusion 4 introduced the <CFLOCK> tag. Using <CFLOCK> it's possible to prevent multiple concurrent access attempts by using code that looks something like this:

```
<CFLOCK NAME="counter" TIMEOUT="10">
  <CFSET counter=counter+1>
</CFLOCK>
```

What's new in CF4.5 locking? Lots, starting with the fact that locking and unlocking of variables is a whole lot faster. In addition, you can now have ColdFusion automatically lock variable read operations (greatly simplifying your code, and preventing possible errors if you by mistake omit a lock, albeit with a performance trade-off). You can also have ColdFusion throw an exception if unlocked variable access occurs (this will help you find potential lock problems and fix them before any damage occurs). And finally, SESSION-variable access can be forced to be single-threaded so that variable access occurs sequentially (further eliminating potential locking problems).



This is only the tip of the locking iceberg, and you should read the updated docs on <CFLOCK> for more information.

CFMAIL Enhancements

This is one we've all been begging for since the day <CFMAIL> was introduced: more control over outbound mail messages. CF4.5 has lots of new features here.

First and foremost, <CFMAIL> now supports blind carbon copy using the new BCC attribute. (I suspect that this single enhancement will dramatically reduce Developer Forum postings!)

Equally important is the addition of the new <CFMAILPARAM> tag. This tag has two uses: mail header manipulation and file attaching. Take a look at this code snippet:

```
<CFMAIL FROM="#email_from#"
  TO="#email_to#"
  SUBJECT="Sales figures">
  <CFMAILPARAM NAME="x-priority"
  VALUE="1">
  <CFMAILPARAM
  FILE="c:\sales\jan.xls">
  <CFMAILPARAM
  FILE="c:\sales\feb.xls">
  <CFMAILPARAM
  FILE="c:\sales\mar.xls">
  Here are the latest sales figures.
</CFMAIL>
```

SCOPE	DESCRIPTION
APPLICATION	Variables shared by all users of a specific application (a single server may have multiple application scopes)
SERVER	Variables shared by all users (all users and applications share the scope)
SESSION	Variables belonging to a single-user session

TABLE 1: ColdFusion persistent variable types

inforum solutions

www.inforumsolutions.com

This outbound mail message uses four <CFMAILPARAM> tags. The first sets a custom message header; the others each attach a file to the message. It's as simple as that.

Process Execution

Several third-party tags are available that can be used to execute applications (or scripts) on your server. Not that there's anything wrong with those tags, but many users cringe at the thought of trusting application execution to unknown third-party code.

Well, cringe no more. CF4.5 now features <CFEXECUTE> as part of the core language. This tag allows you to execute applications synchronously or asynchronously, and even allows you to capture the application output. Look at the following example:

```
<CFEXECUTE
NAME="C:\winnt\system32\ping.exe"
ARGUMENTS="#ipaddress#"
OUTPUTFILE="#temp_file#"
TIMEOUT="30">
```

This code executes the Windows NT-provided ping utility (used to check whether a specific host can be reached). The host address is specified in the ARGUMENTS attribute, and OUTPUTFILE specifies the name of the file to contain the ping output. If you were to read this file, you could use simple string (or RegEx) manipulation functions to check the results.

Improved Debugging

Debugging is something none of us ever want to do, but we end up doing it all the time anyway. So any help in the debugging department is greatly appreciated.

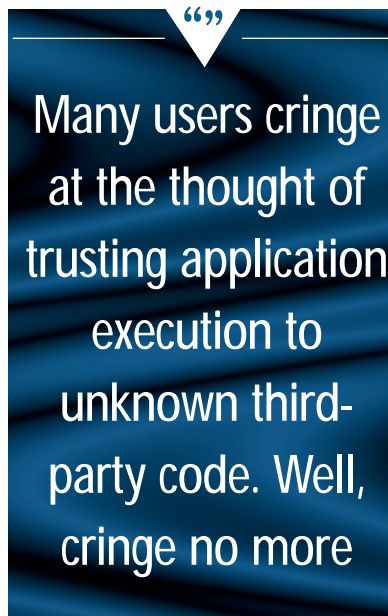
CF4.5 has extended the debug output that gets appended to the end of your pages (if you don't know what I mean, turn on debugging in the ColdFusion Administrator and execute any page on your site). The new information (called "Detail View") provides detailed page execution with individual page components broken out individually. This one is guaranteed to help you pinpoint trouble spots, thereby improving your application's performance.

Access to System Metrics

Continuing with debugging enhancements, say Hello to the new GetMetricData() function (fast becoming one of my favorites). This function returns a

structure containing system performance and activity information. Want to know how many requests are queued? Want to know average request and queue timings? Want to know how fast your database is performing? The function provides access to all that information and more, and all right from within your code.

So what does that do for you? Consider the following. How about intelligently serving live database results if database server activity is low, and serving cached results if the database server is backlogged? Or how about redirection to cached copies of pages if too many requests are queued?



You get the idea. Self-optimizing code – the possibilities are mind boggling.

New Functions

As with every release of CF to date, CF4.5 introduces several new functions (10 of them this time). Included in these are several data conversion functions, the GetMetricData() function I just mentioned, a function that returns a list of all available functions (I'm still trying to find the killer app for this one) and the URLDecode() function.

This last is one I personally have been begging for for years. URLDecode() is the opposite of URLEncodedFormat(); it takes a string that has been URL-encoded and simply decodes it. So why is this so exciting?

For me personally it solves a problem I've been working around for a long time – and it's not a CF problem. It's an HTML limitation problem. If you've

ever tried passing hidden form fields from one page to the next, you'll sympathize with me. HTML form fields have a horrid little limitation – they don't like embedded double quotes (browsers treat these as end-of-field designators). In the past the workaround was to replace the double quotes with other characters or strip them out altogether, neither of which is an ideal solution. The obvious solution was to use URLEncodedFormat() to encode the string (any double quotes would be replaced with %22, which is perfectly safe in a form field), but decoding that string was problematic as CF had no function that did the opposite of URLEncodedFormat(). Not anymore. (Now I need to find time to go back and clean up all that old ugly code.)

White Space Suppression

Okay, those of you who know me know I don't think extraneous white space is a big deal. But judging by the audience reaction when I announced this feature at the User Conference, I must be in the minority here. So here it is.

CF4.5 has added the ability to automatically strip out much of the extraneous white space that used to show up in the generated page output (you know what I mean – all those blank lines you saw when you did a View Source in your browser). You can even turn on a server-wide setting that will automatically perform this suppression for all requested pages served by the server. It's safe, it won't break anything and it will reduce your generated page size slightly.

Conclusion

So there you have it – my list of reasons to upgrade to CF4.5 immediately. If your own list is different, I'd love to hear from you.

The bottom line is that I'm very excited about ColdFusion 4.5. While I don't believe the changes are significant enough to warrant a new book (sorry, had to drop that in there; I get asked about that almost every day), I think you'll find that this new release will make your life just that much easier. And considering how much time we spend at the keyboard, that's a good thing indeed.



BEN@FORTA.COM

ABOUT THE AUTHOR

Ben Forta is Allaire Corporation's Product Evangelist for the ColdFusion product line. He is the author of the best-selling ColdFusion 4.0 Web Application Construction Kit and its sequel, Advanced ColdFusion 4.0 Development (both published by Que), and he recently released Sams Teach Yourself SQL in 10 Minutes.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="intranet">
    <p>
      Intranet
    <select>
      <option onclick="phone">Phone Book</option>
      <option onclick="messages">Messages</option>
      <option onclick="page">Page</option>
    </select>
  </p>
</card>
</wml>
```

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card>
    <p align="center">
      Boston, MA 
    </p>
    <p mode="nowrap">
      <table columns="2">
        <tr>
          <td><i>High</i></td>
          <td><i>Low</i></td>
        </tr>
        <tr>
          <td>53</td>
          <td>42</td>
        </tr>
      </table>
      Today: Mostly sunny and cool.<br/>
      Tonight: Cold with chance of light rain.
    </p>
  </card>
</wml>
```

```
<!-- Get weather info -->
<CFQUERY DATASOURCE="dsn" NAME="weather">
SELECT high, low, image, today, tonight
FROM weather
WHERE city="#city#" AND state="#state#"
</CFQUERY>

<!-- Create output -->
<CFOUTPUT QUERY="weather">
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card>
    <p align="center">
      #city#, #state# 
    </p>
    <p mode="nowrap">
      <table columns="2">
        <tr>
          <td><i>High</i></td>
          <td><i>Low</i></td>
        </tr>
        <tr>
          <td>#high#</td><td>#low#</td>
        </tr>
      </table>
      Today: #today#<br/>
      Tonight: #tonight#.
    </p>
  </card>
</wml>
</CFOUTPUT>
```

The code listing for
this article can also be located at
www.ColdFusionJournal.com

CFWDDAP Writing Custom WDDX Interfaces to LDAP

An implementation of LDAP

BY
ROBERT
FIELDING
&
SARGE
SARGENT

With the advent of Active Directory Services in Microsoft's Windows 2000 and the recent release of Netscape's Directory Server 4.0, the Lightweight Directory Access Protocol is emerging as a mature successor to the X.500 Directory Access Protocol.

More companies are now realizing LDAP's usefulness as the standards-based network access to directory services. Allaire has even integrated LDAP data sources into ColdFusion's Advanced Security infrastructure.

For developers, this means you must add LDAP to your tool set in order to remain marketable. You must familiarize yourself with how LDAP accesses and manipulates data stored in your client's directory server, and you must keep up with the new tools and standards being created to integrate directory services such as DSML (Directory Services Markup Language).

Allaire has given us the CFLDAP tag to integrate ColdFusion with LDAP.

CFLDAP Is Too Simple

The CFLDAP tag is simple to use. For most development purposes the current version of CFLDAP (as of this writing, the current release of CF is 4.01) will get the job done – basic queries, additions, deletions and modifications to directory server entries within a convenient single-tag interface. In the June *CFDJ* (Vol. 1, issue 3) David Anderson's article, "Connecting ColdFusion to LDAP," does a thorough job of explaining CFLDAP basics.

Unfortunately, CFLDAP is too simple for serious LDAP management. Some of its limitations are:

1. **No true record set.** You need a comma-delimited recordset to properly handle multiple group memberships robustly. LDAP queries can return values with commas in the DN, making the returned values ambiguous because you can't differentiate separate entries from entries containing commas.
2. **Commas and semicolons.** CFLDAP properly handles multiple DNs as



well as those containing comma-delimited values (such as addresses); CF (CFOUTPUT) doesn't handle attribute (column) names containing semicolons properly.

3. **Certificates – binary data:** CF's problem with binary data has been documented for a while; this limitation prohibits proper user-certificate management. The X.509 certificate is truncated into a 0-length string since they're binary data. Binary data fields are also named with a syntax that's not friendly to ColdFusion – there's a semicolon in the field name.

The Problem

We've found that most LDAP articles and newsgroup postings (especially Allaire's) focus on the basics of utilizing the LDAP protocol with CFLDAP. In our current task we endeavor to manage and maintain a DOD LDAP, including the enterprise replication of user entries and their PKI certificates. Initially one of our developers created a PERL script to do certificate manipulation because CF

couldn't handle BLOBs (Binary Large Objects), but a true CF resolution was always the goal.

After upgrading to CF 4.x Enterprise edition, we developed a WDDX solution using CFHTTP. However, our entire project is SSL using domestic-strength encryption ciphers and X.509 server and client certificates. Allaire has noted a bug in the WinInet dynamic link library (.dll) that causes https:// calls to fail in CF. In addition to the limitations of CFLDAP listed above, CF's SSL limitation was our biggest hurdle.

To get production back on schedule, coauthor Rob Fielding created a Java HTTP server to handle the limitations of CFHTTP over HTTPS. We then used CFHTTP to post LDAP queries to this Java server. It uses Netscape's ldapsearch function to query our secured LDAP server and responds with a WDDX packet that contains the user's URL-encoded certificate, the user's comma-delimited DN, and the userCertificate;binary and mimeTypeCertificate;binary attribute names.

Note: See "CFHTTP and Secure Sockets Layer" (Article 1096) in Allaire's

A GOOD PLACE TO GET STARTED WITH WDDX IS MATTHEW S. BOLES'S ARTICLE "SERVER TO CLIENT WDDX" IN THE OCTOBER CFDJ (VOL. 1, ISSUE 5).

allaire

www.allaire.com

Knowledge Base for more information on CF's limitation with HTTPS.

Our enigma was how do we remain within our current CF infrastructure but provide user data manipulation in our Netscape Directory Server over SSL. Most developers don't encounter the SSL problem with CFHTTP, so for them just using CFLDAP for their directory server management will suffice. However, we needed a custom solution for this and other related problems, and WDDX was the tool we used. In fact, we'll be writing future articles that deal with custom WDDX and servlet interfaces to LDAPS, JDBC, and more.

WDDX to the Rescue

While participating in the Allaire Forums over the last few months, I've found more of you are running into the same walls we've been driven up by CFLDAP since version 2.0. While it's comforting to finally see others who are feeling our pain, it's more satisfying helping you get around these obstacles. While there are a few custom tags (www.allaire.com/taggallery/) written to extend CFLDAP's functionality, none of them really knock down these walls. To do this we can use the Web Distributed Data eXchange (WDDX) and CF 4.01.

You can use WDDX (XML) as a generic input and output language between servers and clients. Network services, exposed as WDDX servers, no longer need protocol-specific drivers or support for them within CF. Just use Java support to create the WDDX-based interface you'd like to see. Now you can totally change notions about how to deal with LDAPv3 servers to implement a "wish list" of features:

1. All the users get pulled back with their complete DN – including attrib-

utes containing commas – as an actual field in our LDAP object (like CFLDAP).

2. A "group" field containing all group memberships is available for manipulation.
3. I want to be able to extract an LDAP object from a query, modify it and "upload" it back into the server without explicitly having to do a modify statement (ldapmodify). When modifying the "group" attribute, I want it to change my group membership.
4. WDDX packets allow me to manipulate directory data with client technology such as JavaScript and VBScript.
5. Since I must handle binary data, a field ending in ";binary" – such as "userCertificate;binary" or "mimeCertificate;binary" – can be renamed to end in "_binary" inside the WDDX packet. This data can then be placed in URLEncodedFormat. When I go to upload this packet, I'll see that the name ends in "_binary" and realize that I must URLDecode the data and rename the field to end in ";binary" before putting it back into the LDAP. Notice that you can't URLDecode this field in CF since it may contain NULL characters. However, you need to be able to carry it around if you want to pass data between directory servers.

Listing 1 is a query against our XML-based LDAP server. This query wrote an Array called "entryArray" that contains a structure in which every item contains another array containing the actual values. On this point LDAP queries aren't compatible with recordsets. The multivalued attributes are explicitly represented. The returned

packets look similar to Listing 2.

Listing 3 is an example of creating a single LDAP entry from scratch. This entry now exists on the LDAP, and it's in the proper groups. This is much simpler than the way you normally have to deal with CFLDAP. To update one of these entries, you can do the following:

1. Serialize a "backup copy" of the entry to a wddxPacket called "originalEntryPacket."
2. Make whatever modifications you need to the entry. Serialize this modified version of the entry to a wddxPacket called "entryPacket."
3. Call the server again via CFHTTP to send the original and modified packets back to the server, which will compute the intended changes automatically. Since you sent the original version, you're not going to get a conflict if somebody updates your phone number while you update your street. The server will see that the street is different in the two copies and make only the appropriate modifications.

Generalization

This was an implementation of LDAP, but it can be done with other things. WDDX servers make the most sense when you're returning nonrelational data and you'd like to create a language-independent "driver" for a resource that isn't directly supported. As in the case of LDAP, a simple table is insufficient. There are cases where you need full-blown data structures as results (such as WDDX packets representing unformatted reports).



RFIELDING@CPKWEBSE5.NCR.DISA.MIL
SARGENTS@CPKWEBSE5.NCR.DISA.MIL

ABOUT THE AUTHORS

Robert Fielding, the lead Java developer for the DOD GCSS Web/Portal, works for Logicon Information Solutions. He's been using ColdFusion for about a year, and Java since HotJava1.0.

Sarge Sargent, the DBA and lead Web developer for the DOD GCSS Web/Portal, works for Logicon EES. He's been using ColdFusion since 2.0.

LISTING 1

```
<!-- Query the server -->
<cfhttp method=post
url="http://localhost/servlet/LDAPServlet">
  <cfhttpparam type=formfield name=action value=query>
  <cfhttpparam type=formfield name=server value=ws12>
  <cfhttpparam type=formfield name=base value="#UrlEncodedFormat('ou=PKI,ou=DOD,o=U.S. Government,c=US')#">
  <cfhttpparam type=formfield name=filter value="#UrlEncodedFormat('cn=*')#">
  <cfhttpparam type=formfield name=groupbase value="#UrlEncodedFormat('ou=GROUPS,ou=DOD,o=U.S. Government,c=US')#">
</cfhttp>

<!-- Deserialize to a wddxPacket -->
<cfwddx action=wddx2cfml input="#cfhttp.filecontent#"
output='entryArray'>
```

LISTING 2

```
<wddxPacket version='0.9'>
  <header>
  </header>
  <data>
    <array length='2'> <!-- This is an array of LDAP entries -->
      <struct> <!-- This is the first LDAP entry -->
        <var name='cn'>
          <array length='1'>
            <string>Smith.Joe</string>
          </array>
        </var>
        <var name='mail'>
          <array length='2'>
            <string>Smith.Joe@foo.com</string>
```

LISTING 3

```

entry.sn=ArrayNew();
entry.sn[1]="Smi th";
entry.uid=ArrayNew();
entry.uid[1]="99000004321";
entry.groups=ArrayNew();
entry.groups[1]="cn=ADMIN,ou=GROUPS,ou=DOD,o=U. S.
Government,c=US";
entry.groups[2]="cn=WEB,ou=GROUPS,ou=DOD,o=U. S.
Government,c=US";
</cfscript>
<!-- Serialize to a wddxPacket -->
<cfwddx action=cfml2wddx input=' #entry#'
output='wddxEntry' >
<!-- Send it to the server -->
<cfhttp method=post
url ="http://localhost/servlet/LDAPServlet">
<cfhttpparam type=formfield name=server value=ws12>
<cfhttpparam type=formfield name=action value=add>
<cfhttpparam type=formfield name=username
value="#UrlEncodedFormat('cn=Directory Manager')#">
<cfhttpparam type=formfield name=password
value=XXXXXX>
<cfhttpparam type=formfield name=entry
value="#UrlEncodedFormat(wddxEntry)#">
</cfhttp>

```

The code listing for
this article can also be located at
www.ColdFusionJournal.com

ColdFusion with Classes

Construct complex enterprise systems that perform

BY RALPH FIOL

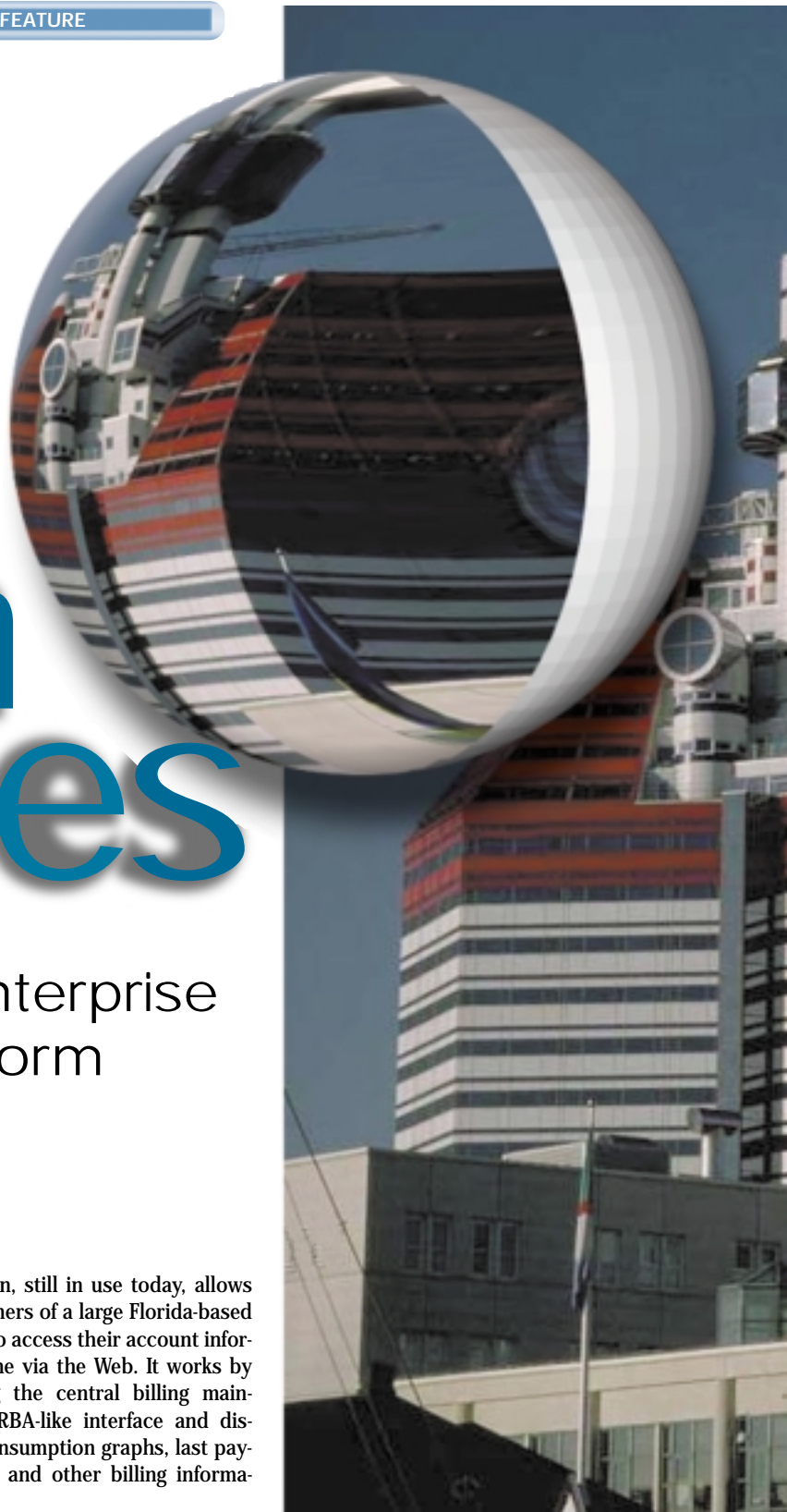
Just over four years ago I completed what I now consider my first “significant” Web-based application. By *significant*, I mean the project had a significantly large budget, a significantly large user audience and a significantly high level of complexity.

The application, still in use today, allows residential customers of a large Florida-based power company to access their account information in real time via the Web. It works by directly querying the central billing mainframes via a CORBA-like interface and displaying energy consumption graphs, last payment information and other billing information.

Looking back, I’m amazed at how much the Web development tools and application servers have evolved and morphed into the scalable, robust, full-featured products they are today. Back then, ColdFusion was scarcely known, and ASP didn’t exist (ISAPI and IDC were it). And yet, as much as I love the elegance and power of today’s ColdFusion, I often crave the features found

in the tools we used in those early days.

Let me explain. For this “significant project” I used a tool called VisualWave made by what was then ParcPlace-Digital (acquired by ObjectShare, which later sold the product to Cincom). VisualWave is a Smalltalk-based programming tool that’s 100%, no-getting-around-it, hard-core object-oriented. For the





sion as opposed to program by reinvention.

The problem, of course, was that it suffered from tremendous performance issues. VisualWave was ParcPlace's attempt to take their flagship product, VisualWorks – a wonderful, single-user, single-threaded (some will argue otherwise) tool – and “Webify” it.

Recently I began thinking, Wouldn't it be awesome if ColdFusion – a powerful, tag-based, easy-to-use, highly scalable product – supported the use of objects – inheritance, polymorphism, encapsulation – the whole shebang? And wouldn't it be cool if this could be implemented without degrading performance at all (or at least only minimally)? As I considered the magic of this, it occurred to me that C++ evolved from similar objectives. C++, originally called C with Classes, has all the benefits of a procedural language (performance, efficient memory utilization, low system resource requirements, etc.) and the elegance of objects and classes. That's what I wanted – CF++ – ColdFusion with Classes! As a self-proclaimed object guru and ColdFusion maharishi, I decided it was my responsibility to develop an object-based framework for ColdFusion.

The Implementation

I came up with CFC, a simple, elegant and efficient framework for implementing inheritance, polymorphism and encapsulation with CFML. The framework uses advanced ColdFusion 4.01 features such as exception handling, collections and the request scope to satisfy many of the requirements. Until 4.01, the implementation of this framework wouldn't have been practical.

At the core of CFC are four custom tags:

1. **CreateObject**
2. **InvokeMethod**
3. **CollectGarbage**
4. **CopyObject**

CreateObject

CreateObject expects two attributes: CLASS and OBJECT. Here's an example of how to invoke it:

```
<cfmodule name="CFC.CreateObject"
  class="Contact"
  object="aContact">
```

The attribute CLASS names the class definition or template (more on this shortly). The attribute OBJECT names the output variable to be created for the object instance. Thus,

after this custom tag is invoked, there'll be a local variable named “aContact” representing the object.

CreateObject will automatically call the constructor method for each object created. If implemented, this method allows the developer to include any special initialization code that may be required for the object. Additionally, CreateObject will save a reference to the object in a global (Request scope) collection called the “Object Pool,” which is used by the garbage collection mechanism invoked during OnRequestEnd.cfm.

Each CFC class is defined by creating a file directory with the same name as the class. Thus, in our example above, there must be a directory named “Contact,” which resides under the Object Library Directory (see Figure 1). The Object Library Directory for a particular application is defined by a ColdFusion Mapping.

A special file named class.cfm must exist within the class directory. This file contains the name of the class and its superclass. CreateObject will use this information when creating each object, loading each class.cfm for all superclasses so as to construct a class hierarchy. Later, when methods are invoked, the CFC framework will use the class hierarchy information stored within each object instance to determine where a method is implemented (base class, superclass, etc.).

Here's an example of the class.cfm file for the class Contact.

```
<cfscript>
variables.className = "contact";
variables.superClass = "object";
</cfscript>
```

The CreateObject custom tag will return an object instance. Each instance is simply a ColdFusion structure that contains the class name, superclass hierarchy, instance ID and any additional attributes defined by the application. Once created, an object can be used like any other structure. In other words, to set attributes simply issue CFSET statements as this example demonstrates:

```
<CFSET aContact.FirstName = "Ralph">
```

Attributes can also be defined at create time by passing them as attributes to the CreateObject tag. For example:

```
<cfmodule name="CFC.CreateObject"
  class="Contact"
  object="aContact"
  FirstName="Ralph"
  LastName="Fiorio">
```

InvokeMethod

After instantiating an object, you can invoke any of its methods using the CFC

developer, VisualWave was a delight. (For the Webmaster, let's just say you couldn't pay me enough to do that job!) VisualWave allowed us to use objects and classes to model components of the real world (a customer, a bill, a payment, etc.). We could assign both attributes and behaviors to these objects, and use inheritance to program by exten-

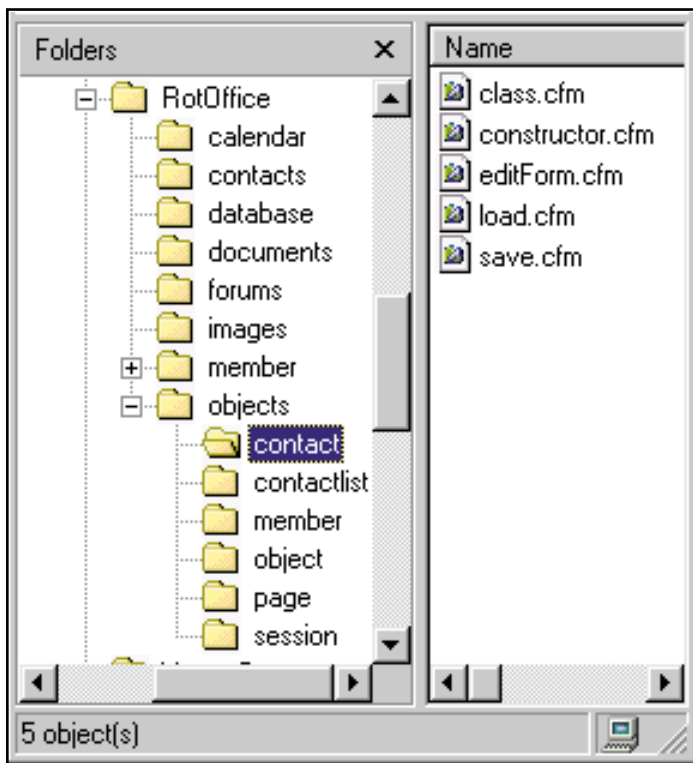


FIGURE 1: The Contact class files, which reside under the Object Library Directory

Framework custom tag named (you guessed it) `InvokeMethod`. Here's an example of how to call it.

```
<cfmodule name="CFC.InvokeMethod"
  object="#aCustomer#"
  method="loadFromCookie">
```

The attribute `OBJECT` must reference a structure created by a call to `CreateObject`. The attribute `METHOD` must name a method implemented by the class or any superclass of the object. Additional attributes may be passed to `InvokeMethod` – they'll be forwarded to the named method. Methods are implemented as custom tags residing within the class directory (see Figure 1). Here's how it works.

`InvokeMethod` will begin by attempting to call a custom tag named by the `METHOD` attribute. It'll assume that the custom tag for the method exists in the immediate class directory. Since this isn't a requirement, it wraps the `CFMODULE` call within a `CFTRY/CFCATCH` exception block. If an exception is detected, it'll work its way up the class hierarchy and attempt the `CFMODULE` again for the superclass. This will continue all the way up to the root (base) class. If the method isn't found within any class in the hierarchy, `InvokeMethod` will throw an exception to the caller.

`InvokeMethod` will pass all attributes to the called method within a structure called `msgArguments`. Additionally, it'll pass a special attribute named `"self"` to the method. `Self` is a reference to the object instance (the structure) and is useful for the implementation of each method. In the RotOffice sample application discussed later, the base class named "Object" contains a method called `dump` (see Figure 2). This method simply outputs its instance variables for debugging purposes, using the attribute `self` to refer to its instance variables. Here's a snippet of that method.

```
<cfoutput>
  ClassName: #attributes.self.ClassName# <br>
  SuperClass: #attributes.self.SuperClassName# <br>
  InstanceID: #attributes.self.InstanceID# <br>
</cfoutput>
```

CollectGarbage

`CollectGarbage` simply iterates over the Object Pool and sends the destructor method to each of the objects created during an HTTP request. It's designed to be called by the application's `OnRequestEnd.cfm` file. This is useful for objects like the Session object implemented in RotOffice (see Figure 2). The session object updates its `HitCount` and `LastVisit` attributes, then stores itself as a cookie (a WDDX packet) during its destructor method. Other objects may choose to store themselves to a database, update log files, and so on. (Note: The Session object implemented in the sample application is a great alternative to the ColdFusion session implementation. As you know, session variables can't be used in a clustered environment since ColdFusion stores session information in memory on the Web server [which isn't shared]. The Session object implemented here stores itself on the client machine and thus is made available to all machines in the cluster.)

CopyObject

The final custom tag in the framework is simply a convenience tag. It does a shallow copy of all attributes from one object to another. This is useful for copying the attributes of a contact object to a session object, for example.

A Sample Application

The name of the accompanying sample application titled "RotOffice" is, as you may have guessed, a rip-off of the site HotOffice.com, a wonderful virtual office and collaboration site. For demonstration purposes RotOffice implements a basic authentication system based on the Session object introduced earlier and a simple Contact Management feature. This sample application can be downloaded from CFDJ's Web site (www.ColdFusionJournal.com).

To construct this application I've implemented several classes. At the root of the class hierarchy is `Object`. `Contact` and `Session` inherit

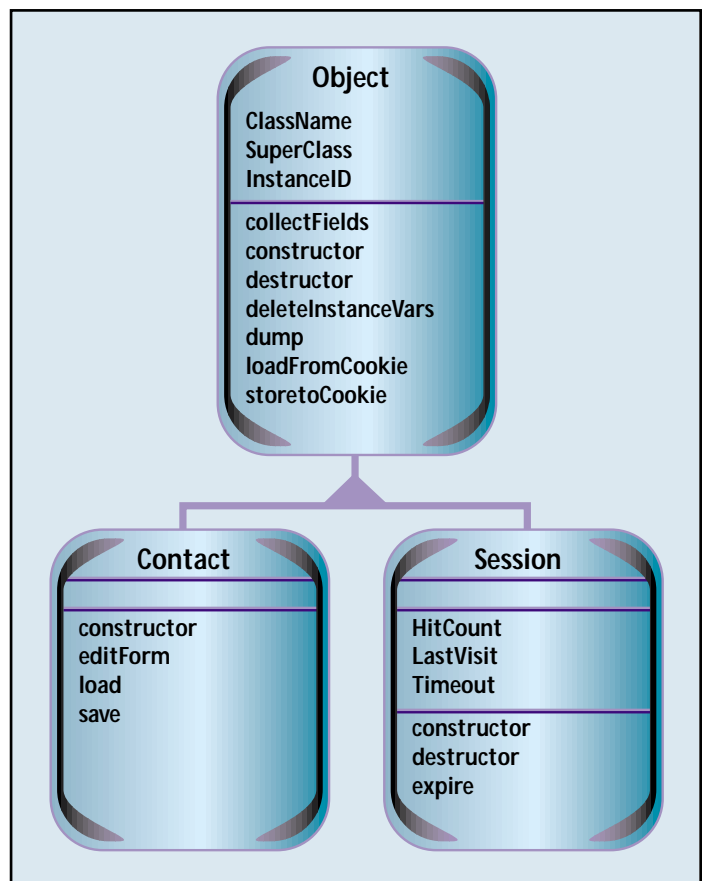


FIGURE 2: Some of the objects implemented in RotOffice sample application

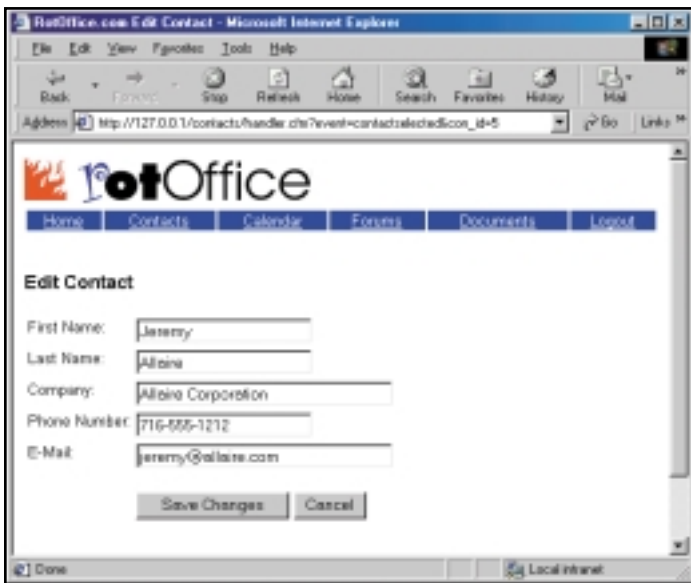


FIGURE 3: An HTML form for a contact, with pre-filled fields of instance variables

from this class as shown in Figure 2, as do Page, ContactList and Member.

You'll notice that the CFC Framework attempts to separate what I'll refer to as "views" of an application from the actual objects. A *view* is simply a CFML page. In the sample application you'll notice a directory under \objects for each of the classes and, in most cases, a corresponding directory under the RotOffice Web root for each view. An important component of the framework is that views respond to events, and they use objects to implement the appropriate responses to these events. You'll notice a special file called handler.cfm under

each view's directory, along with a file for each possible event. The default event for each view (as defined in handler.cfm) is, conveniently, "index." In our example the view named Contacts also has a handler for the events OnAdd, ContactSelected and ContactChanged.

The complete source for the event index (implemented in index.cfm) of the view Contacts is given in Listing 1.

And it's short and sweet! We begin by creating an object, aPage, that generates the HTML necessary for the pages (views) in our application. During its constructor event aPage generates the HTML header and body tags. During its destructor it generates the end tags.

Next we construct a ContactList object. This class implements several methods for accessing and displaying contacts within our database. Several display formats can be defined, and each will ultimately list the contacts in an HTML table. The method shown here – loadAndDisplay – presents the contacts in a table and makes each name clickable. The HREF value for the clickable text looks something like this:

```
<a href="handler.cfm?event=contactSelected&con_id=..." >
```

Notice that if a user clicks on a contact within this view, the handler file will be requested and the URL.Event will be set to contactSelected. The handler.cfm file will assume that there's a corresponding file for that event named contactSelected.cfm and will CFINCLUDE it. If it's not found, handler.cfm will reroute the event to the root handler or handle the event based on the requirements of the specific application. The source code for the view contactSelected.cfm appears in Listing 2.

As in the previous example, we begin by creating a page object. It should be noted here that during its constructor event the page object automatically invokes a method, collectFields, that converts all URL and FORM variables submitted into instance variables of the object. If you're familiar with Fusebox (www.fusebox.org), this is similar to the "FormURL2Attributes" tag. In our source code this is used during the

sitehosting.net

www.sitehosting.net

load method of the aContact object. It's important to us because our application doesn't have to be concerned about whether this view is an action page of a form or the recipient of URL parameters. The page object abstracts that for us, converting the URL.con_id parameter to the instance variable aPage.con_id.

Last, we invoke the method editForm of the object aContact, which will display an HTML form for a contact, prefilling all fields with the instance variables if available (see Figure 3). It'll post back to handler.cfm with an event of contactChanged if the user clicks the submit button. This same form can be used for adding and updating a contact. Let's take a look at the handler for contactChanged. Here's the complete source:

```
<cfmodule name="CFC.CreateObject"
  class="contact"
  object="aContact">

<cfmodule name="CFC.InvokeMethod"
  object="#aContact#"
  method="collectFields">

<cfmodule name="CFC.InvokeMethod"
  object="#aContact#"
  method="save">

<cflocation url="index.cfm">
```

First, we instantiate a contact object and invoke the method collectFields. As with the page object, this method converts URL and FORM fields into instance variables of the receiver object. The method is implemented by the class Object and is therefore inherited by both Page and Contact (and all other subclasses!). Finally, we simply invoke

the method save, which writes the instance variables to the database, and show the contact list.

The ContactList Class

As you can see from the source listings thus far, little actual code is found in the views. Most of the real work is implemented in the methods of the classes Page, Contact and ContactList. To illustrate how some of this is implemented, let's look at the class ContactList. You'll recall that we invoked the method loadAndDisplay during the index event of our contacts view. This method simply queries the database to load all contacts, then displays them in a table. Here's the source code for the method.

```
<cfmodule name="CFC.InvokeMethod"
  object="#attributes.self#"
  method="loadAll"
  offi ce="#attributes.msgArguments.offi ce#">

<cfmodule name="CFC.InvokeMethod"
  object="#attributes.self#"
  method="display">
```

Seems reasonable. First we invoke the loadAll method to load all contacts from the database into self, the object's instance structure (notice the use of attributes.self). Then we invoke the display method to generate the table. The two processes – load and display – are implemented separately so as to provide a more flexible and useful object. Thus the loadAndDisplay method is simply a convenience method. The loadAll method is implemented as:

```
<cfquery name="attributes.self.contacts"
  datasource="#application.db_dsn#">
```

What Is Object-Oriented Programming?

Object-oriented programming is a methodological framework for software engineering. By providing support for the objects and classes of an application domain, the object-oriented paradigm creates a better modeling and implementation of systems. Objects provide a focus throughout analysis, design and implementation by emphasizing the state, behavior and interaction of objects in its models.

The object-oriented, problem-solving approach is similar to the way a person solves daily problems. It consists of identifying objects and using them in the correct sequence to solve the problem. In other words, object-oriented problem solving can consist of designing objects whose behavior solves a specific problem. A message to an object causes it to perform its operations and solve its part of the problem.

Terms

- **Class:** A specification of structure (data), behavior (methods) and inheritance (parent classes) for objects.
- **Object:** A single object is simply an instance of a class. An object can be considered a "thing" that can perform a set of activities that defines the object's behavior. For example, a "Student" object can tell you its grade point average, year in school, its name or its address. The object's interface consists of a set of commands, each one performing a specific action. An object asks another object to perform an action by sending it a message. The requesting (sending) object is the sender; the receiving object, the receiver.

- **Inheritance:** Inheritance provides a natural classification for kinds of objects. It's a relationship between classes in which one class is the parent (base/superclass/ancestor) class of another. Inheritance provides programming by extension rather than reinvention. It allows us to reuse existing classes by making a small change to a class – for example, creating a subclass to alter a method or adding a method to a parent class. With differential programming, a class doesn't have to be modified if it's close to what's required; a derived class can be created to specialize it. This avoids code redundancy; otherwise code would have to be copied and modified.

Inheritance provides for code and structural reuse. This kind of reuse takes advantage of the "is-a-kind-of" relationship. Class libraries allow reuse between applications, potentially allowing order-of-magnitude increases in productivity and reductions in defect rates (program errors), as library classes have already been tested and further use provides further testing, providing even greater reliability.

- **Method:** A method implements behavior, which is how an object acts and reacts in terms of its state changes and message passing. A method is a function or procedure that's defined in a class and can typically access the internal state of an object of that class to perform some operation. It can be thought of as a procedure, with the first parameter as the object to work on. This object is the receiver, which is the object the method operates on.

sd 2000

www.sdexpo.com


```
select      contact.*
from        contact
where       contact.off_id =
#attributes.msgArguments.offidce#
order by
    contact.con_lname,
    contact.con_fname
</cfquery>
```

Summary

ColdFusion provides us with a first-class RAD (Rapid Application Development) environment for the construction and deployment of Web applications, and its performance, universal database connectivity and extensibility provide us with the means to construct scalable applications quickly.

Combine the two and we have “ColdFusion with Classes,” a toolset that makes it possible to construct complex enterprise systems that

Perhaps Allaire will add a “Class Gallery” to complement the Tag Gallery on their site.

Now then, who's going to build a class hierarchy browser? 

1. Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*. 2nd ed. Addison-Wesley.
2. Goldberg, A., and Robson, D. (1989). *Smalltalk 80: The Language*. Addison-Wesley.
3. Rumbaugh, J., et al. (1991). *Object-Oriented Modeling and Design*. Prentice Hall.
4. Wirfs-Brock, R., Wilkerson, B., and Wiener, L. (1990). *Designing Object-Oriented Software*. Prentice Hall.

Ralph Fiol is president of Media XXI Group Corporation, a consulting firm based in Miami, Florida, that specializes in e-commerce and systems-integration services. Ralph is a certified ColdFusion Instructor and author of Visual C++ MasterClass 4 (Wrox Press).

ralph@fiol.net

```
<cfmodule name="CFC.CreateObject"
    class="page"
    object="aPage"
    title="Contacts">

<cfmodule name="CFC.CreateObject"
    class="contactList"
    object="aList">

<a href="handler.cfm?event=OnAdd">
Add a Contact
</a>

<cfmodule name="CFC.InvokeMethod"
    object="#aList#"
    method="loadAndDisplay"
    offline="#aSession.off_id#">
```

```
<cfmodule name="CFC.CreateObject"
    class="page"
    object="aPage"
    title="Edit Contact">

<cfmodule name="CFC.CreateObject"
    class="contact"
    object="aContact">

<h1>Edit Contact</h1>

<cfmodule name="CFC.InvokeMethod"
    object="#aContact#"
    method="editContact">
```

```
<cfmodule name="CFC.InvokeMethod"
    object="#aContact#"
    method="editForm">
```

<div></div>	
Last Name	
First Name	
Company	
Phone	
E-Mail	

<cfoutput query="attributes.select contacts">

<tr valign="top">

<td>#attributes.select.contacts.con_lname#</td>

<td>#attributes.select.contacts.con_fname#</td>

<td>#attributes.select.contacts.con_company#</td>

<td>#attributes.select.contacts.con_phone1#</td>

<td>#attributes.select.contacts.con_email#</td>

</tr>

</cfoutput>

</table>

DDDDDDDDDD

The code listing for
this article can also be located at
www.ColdFusionJournal.com

allaire

www.allaire.com

Even Web spiders are afraid of something

Virtual Arachnophobia

BY JOHN MORGAN

There once was an emporium that sold the finest treasures in all the world. Their service was second to none, and their prices were more than reasonable.

But they had a problem. The problem was that only a few knew where the emporium was, and most didn't even know it existed. Sadly, the emporium failed because their potential customers couldn't find them.

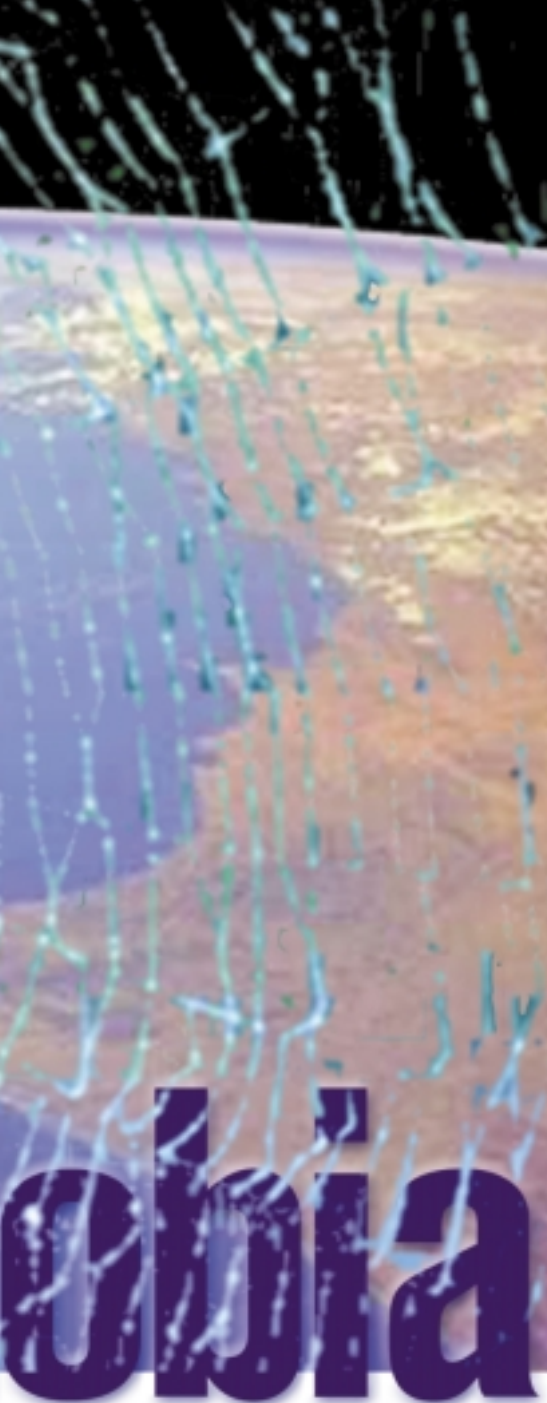
A similar tale of woe could be told of many would-be-great Web sites. Web sites destined to fail because they ignore the first rule of retail success: "Location, location, location." What is true in the physical world is even truer of the virtual world. If your customers can't find you, they're not your customers.

This story was also true for me. I had created a dynamic, database-driven Web site for Blue Star Training, the company I work for. I took great pains to get it listed with all the

best search engines. So I was shocked to learn that the "meat" of my Web site, the dynamic pages, weren't indexed by the search engine's spiders.

The Problem with Spiders

Web spiders are systems designed to crawl through a Web site and index the content and its links. Many search engines launch a spider when a request is made to index a site. Most spiders will happily follow all links except those that contain a question mark. Did you catch that exception? That exception means that pages with URLs that pass parameters are not indexed. This is a serious problem for ColdFusion sites because dynamic URLs are the lifeblood of a dynamic site.



course information pages, were not indexed by the spiders. The summary page was indexed, but that's not enough to ensure a good location in the search engine listings.

Ben Forta Offers a Solution

Like any smart ColdFusion developer, I went to the Developer Support Forums on Allaire's Web site to see if anyone had already solved my problem. Eureka! I found a solution written by Ben Forta (http://forums.allaire.com/DevConf/Index.cfm?Message_ID=18401).

Ben's solution was to replace the question mark with a forward slash and list any parameter values like this:

```
<A HREF="CourseInfo.cfm/HT1">
```

This would cause the spiders to see the URL as a static link and follow it. The next step would be to parse CGLPATH_INFO to extract the value being passed.

I Still Have Problems

Ben's solution was good, and it worked, but it presented a few problems for me. The first problem was that my image tags like this

```
<IMG SRC="images/Logo.gif">
```

looked for the image in the path /CourseInfo.cfm/images/, which of course didn't exist. This behavior was caused by the browser mistaking CourseInfo.cfm for a directory. This was easily overcome by the use of <BASE HREF="http://www.bluestarcorp.com/">.

The second problem wasn't so easy. I had my IIS server set to check for the existence of every .cfm file before passing the page request to ColdFusion. I did this so I could catch "404 Not Found" errors and display a custom error page.

In case you didn't know it, IIS passes requests for all .cfm files to the ColdFusion server, and it's ColdFusion that displays the error if the file can't be found. The error message that ColdFusion displays is an ugly and unfriendly one that you can't modify. To prevent this you need to tell IIS to check whether the file exists before the request gets passed to ColdFusion, and let IIS route browsers to a custom error page.

Trapping the 404 Errors

Let me tell you how I trap the "404 Not Found" error, because it'll be important later. From the Microsoft Management Console in IIS you select your site in the left panel and right-click and select "Properties". Next, click on the "Home Directory" tab and the "Configuration" button (see Figure 1). Locate and select the .cfm application mapping in the list (see Figure 2). Click the "Edit" button to display the mapping properties. Finally, click the "Check that file exists"

checkbox (see Figure 3) and then click the "OK" button.

Now IIS will check for 404 errors, but you still need to tell IIS what page to display when a 404 error occurs. Back in the Web site properties you click on the "Custom Errors" tab. Locate and select the 404-error entry from the list (see Figure 4) and click the "Edit Properties" button. With the "Message Type" drop-down list select URL. In the "URL:" text input, type the URL for your custom 404-error page (see Figure 5). Finally, click the "OK" button to close the dialog box and then click the "OK" button to close the properties window.

How I Found a Solution by Solving Another Problem

I wasn't having any luck finding a solution to the dynamic URL problem, so I began working on another problem I had. The problem was that I had converted www.bluestarcorp.com from HTML to CFML, and I had old entries in the search engine listings that still pointed to the old HTML pages. I still wanted to get the traf-

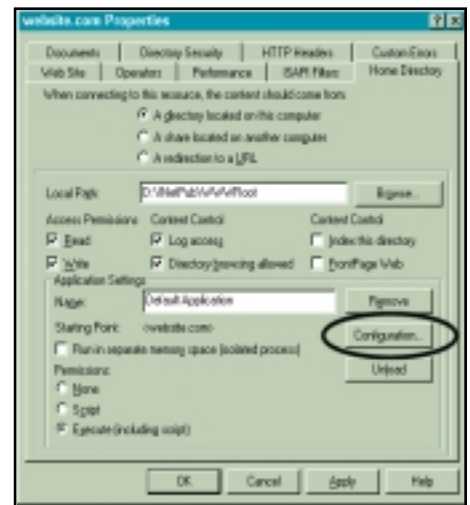


FIGURE 1: IIS Web site properties

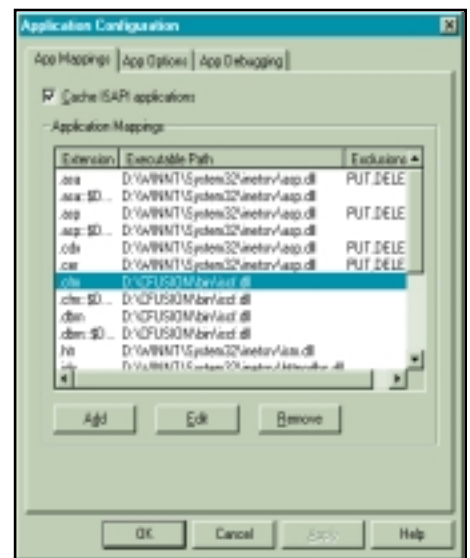


FIGURE 2: IIS application mappings

I Don't Know What's Worse, the Bugs or the Spiders

I was suffering from the exact problem stated above. I had designed a catalog of courses for our IIS-hosted ColdFusion Web site. It had two main pages, the page for displaying short summaries of each course in the catalog and the page that displayed the details about a given product.

The summary page had a link for each product that took you to the detail page. The link for a given course looked something like this:

```
<A HREF="CourseInfo.cfm?COURSE_ID=HT1">
```

Did you notice the question mark? The pages that I needed indexed the most, the

fic from the old listings, but I didn't want to have to do a lot of work.

Helping 'Broken' URLs Find the Way Home

That's when the thought hit me: all traffic to my site that is for a nonexistent page goes through the custom 404-error page. Since I made my custom error page a ColdFusion template, I could check what page the browser was requesting and route it to the corresponding CFML page. If the browser requested calendar.html, I'd route them to calendar.cfm.

So I wrote a little code to test if the requested file name ended in .html (see Listing 1). If it did, I replaced the .html with .cfm and used <CFLOCATION> to take the browser to the correct page; if it did not, I displayed the error message.

You might be wondering, What if a request is made for an HTML file that doesn't have a corresponding CFM file? Well, in that case the code would route the browser to a nonexistent CFM page and would end up back on the 404-error page and finally get the 404-error message. I was examining my 404 handling code one day when the second idea hit me.

The Plan of Attack

I reasoned that if I could parse the URL and check for .html from the 404-error page, I

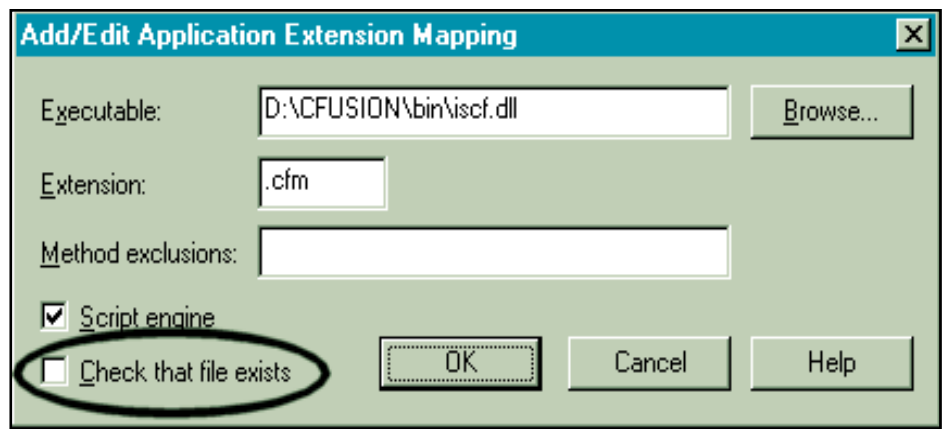


FIGURE 34 Make IIS check that the file exists

could parse the URL for anything. Here's the idea: create a link to a page that doesn't exist, then parse the URL to get to the desired page and pass any parameters. The transformed URL would look like:

```
<A HREF="DYN_HT1.cfm">
```

Once more I wrote a little code, this time to check whether the requested file name started with "DYN_" (see Listing 2). This is my signal that the URL is not real but must be parsed. To parse out the parameter (in this case my course code), I locate the first character after the "DYN_" marker and the last character before the .cfm extension

and store all characters between these points in a local variable. Following the example so far, that would mean that I would store HT1 (my course code) in that local variable.

Now the plan was to use <CFLOCATION> to route the browser to the appropriate page along with the question mark and the parameters. Well, it worked, but the spiders don't look just at the original link address, but also at the address reported in the resulting page's HTTP header. Because I used a <CFLOCATION>, the URL in the HTTP header would be "CourseInfo.cfm?COURSE_ID=HT1". The spider once again refused to index my page.

virtualscape

www.virtualscape.com

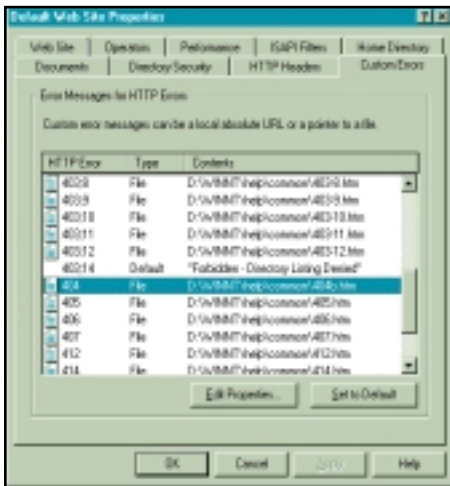


FIGURE 4 Custom error files

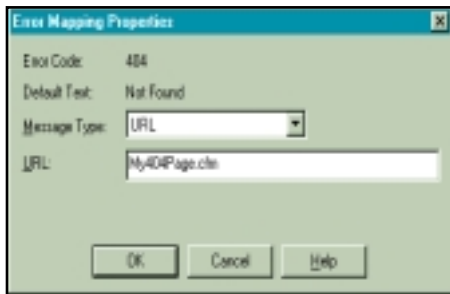


FIGURE 5 Custom error properties

What the Spider Doesn't Know...

I was at the end of my rope when I remembered one of ColdFusion's greatest strengths, the `<CFINCLUDE>` tag.

My thinking was, if I `CFINCLUDE` the contents of the desired page (the details page in this case) into the 404-error page, the spider would never know it. Now you need to know that when the 404-error page is sent back to the browser, the originally requested URL is the one written to the HTTP headers. That means that the spider will see the desired page but will "think" it's coming from the static URL.

One Last Hurdle

I went back and modified my code to `CFINCLUDE` the "CourseInfo.cfm" template along with the desired course ID (see Listing 3). Guess what?...You guessed it. It didn't work. The `CFINCLUDE` will not take URL parameters. It makes sense that `CFINCLUDE` won't take parameters because the template in question is not being evaluated but is being merged into the 404-error template.

I had come too far to let this stop me. I knew that you could create form variables on the fly. So, I reasoned, why not URL variables?

I went back yet again and modified my code, this time to create a URL variable and then `CFINCLUDE` the "CourseInfo.cfm" template (see Listing 4).

Eureka once again! It worked! I could now use the URL "DYN_HT1.cfm" and get to the same place as if I had used

"CourseInfo.cfm?COURSE_ID=HT1".

I went back and modified my summary page to generate links in the DYN_COURSEID format for each course as it was read from my database. I tested several of my links. Then – with a little bit of trepidation – I submitted the catalog summary page to several search engines.

I'm happy to say that my pages indexed with every search engine I submitted them to. The spiders never knew that the pages they indexed were dynamically generated.

How It All Works

Let's recap how this technique, which I call "Faking a URL," works (see Figure 6). The page request comes from the browser in the form "DYN_HT1.cfm". IIS tests to see if

"DYN_HT1.cfm" file exists, which it doesn't. Because it doesn't, IIS routes the browser to the 404-error page. The page "sees" that the file name starts with "DYN_" and parses out the course ID parameter (HT1 in this case). The 404-error page code `CFINCLUDEs` the CourseInfo.cfm template. The CourseInfo.cfm template code reads the value in URL.COURSE_ID and displays the correct content for the HT1 course.

The Right Tool for the Right Job

You shouldn't use the "fake URL" technique to trick the spider for just any dynamic page. It's best suited for dynamic pages that display mostly database output, like my course catalog. The catalog information doesn't change very often, but often enough that I

digital nation

www.dedicatedserver.com

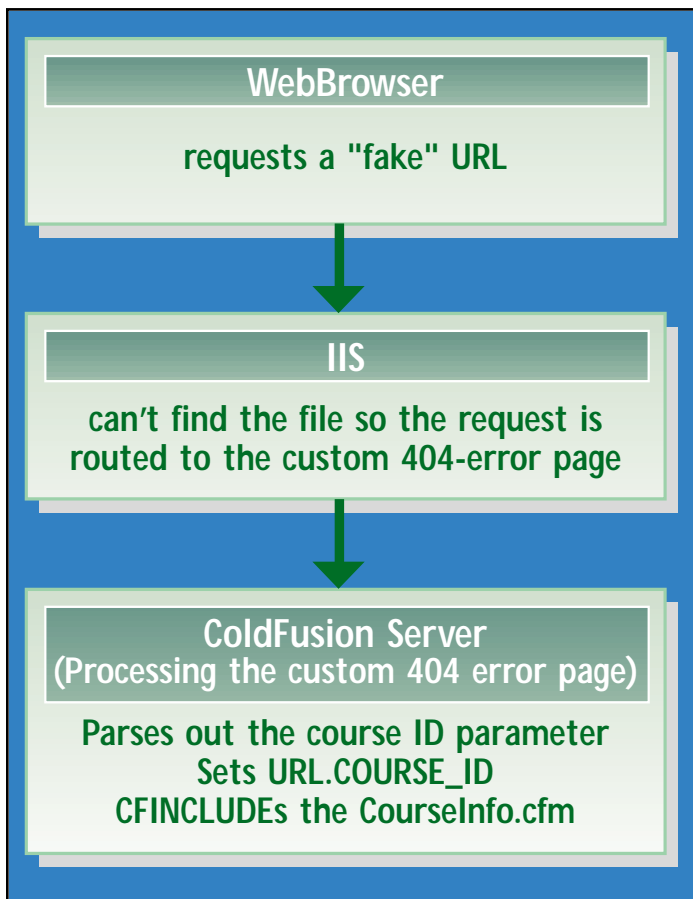


FIGURE 6: The "fake" URL system

wouldn't want to "hard-code" my pages. If I make major additions or changes to my course catalog, the spider will pick them up on the next visit.


Keep in mind that you still have the flexibility of linking to the dynamic page with the old method via a URL like "CourseInfo.cfm?COURSE_ID=HT1".

This was a huge timesaver for us because we have links to our courses all over the Web site. But we only had to change the links on the catalog page to account for the spiders. We don't care that all the other dynamic links are ignored, because to spider them would be redundant.

Extending the 'Fake URL' Technique

You can add additional blocks of code to the 404-error template to process different prefixes. We have a second prefix, "BIO_", that we use for instructors. You can add as many prefixes as you need.

You can also extend the code to accept multiple parameters. Thus, if you wanted to have one page for sale items costing less than \$10, you could use a URL that looks like "ITM_CATEGORYID-10.cfm" to display the list "bargain items". The best way would be to treat the parameter list like a ColdFusion list, with a dash as the delimiter, and then use CFLOOP to "break out" each of the parameters for use.

Use your imagination and see what you can come up with. If you find an interesting way to extend the fake URL system, I'd love to hear from you. 

About the Author

John Morgan is busy training programmers, Web developers and database developers as well as writing courseware at Blue Star Training. He speaks at conferences, workshops and the San Diego ColdFusion Users Group, which he hosts.

JohnMorgan@bluestarcorp.com

LISTING 1

```

<!-- Find the end of the protocol -->
<CFSET URLStart = (FindNoCase("/", CGI.QUERY_STRING))>
<CFIF URLStart IS NOT 0>
    <!-- Remove information before the URL -->
    <CFSET Target = RemoveChars(CGI.QUERY_STRING, 1, (URLStart - 1))>
<CFELSE>
    <!-- This will force standard processing -->
    <CFSET Target="Could not parse">
</CFIF>
<CFIF FindNoCase(".html", Target) EQ 0>
    <HTML>
    <HEAD>
    <TITLE>Error 404</TITLE>
    </HEAD>
    <BODY>
    <!-- Error Message -->
    <H1>ERROR: The requested page could not be found.</H1>
    </BODY>
    </HTML>
<CFELSE>
    <CFSET CFMTTarget = Replace(Target, ".html", ".cfm", "ONE")>
    <CFLOCATION URL="#CFMTTarget#">
</CFIF>
  
```

LISTING 2

```

<!-- Find the end of the protocol -->
<CFSET URLStart = (FindNoCase("/", CGI.QUERY_STRING))>
<CFIF URLStart IS NOT 0>
    <!-- Remove information before the URL -->
    <CFSET Target = RemoveChars(CGI.QUERY_STRING, 1, (URLStart - 1))>
<CFELSE>
    <!-- This will force standard processing -->
    <CFSET Target="Could not parse">
  
```

```

</CFIF>
  
```

```

<!-- Check for dynamic course info code -->
<CFSET DynamicStart = FindNoCase("DYN_", Target)>
<CFIF DynamicStart IS NOT 0>
    <CFSET DynamicStart = DynamicStart + 4>
    <CFSET DynamicEnd = FindNoCase(".", Target, DynamicStart)>
    <CFSET Target = Mid(Target, DynamicStart, (DynamicEnd - DynamicStart))>
    <CFLOCATION URL="CourseInfo.cfm?URL.COURSE_ID=#Target#">
    <CFEXIT>
</CFIF>

<CFIF FindNoCase(".html", Target) EQ 0>
    <HTML>
    <HEAD>
    <TITLE>Error 404</TITLE>
    </HEAD>
    <BODY>
    <!-- Error Message -->
    <H1>ERROR: The requested page could not be found.</H1>
    </BODY>
    </HTML>
<CFELSE>
    <CFSET CFMTTarget = Replace(Target, ".html", ".cfm", "ONE")>
    <CFLOCATION URL="#CFMTTarget#">
</CFIF>
  
```

LISTING 3

```

<!-- Find the end of the protocol -->
<CFSET URLStart = (FindNoCase("/", CGI.QUERY_STRING))>
<CFIF URLStart IS NOT 0>
    <!-- Remove information before the URL -->
    <CFSET Target = RemoveChars(CGI.QUERY_STRING, 1, (URLStart - 1))>
<CFELSE>
    <!-- This will force standard processing -->
  
```

piranhane

www.piranhanet.com

```
<CFSET Target="Could not parse">
</CFIF>

<!-- Check for dynamic course info code -->
<CFSET DynamicStart = FindNoCase("DYN_",Target)>
<CFIF DynamicStart IS NOT 0>
    <CFSET DynamicStart = DynamicStart + 4>
    <CFSET DynamicEnd = FindNoCase(".",Target,DynamicStart)>
    <CFSET Target = Mid(Target,DynamicStart,(DynamicEnd - DynamicStart))>
    <CFINCLUDE TEMPLATE="CourseInfo.cfm?COURSE_ID=#Target#">
    <CFEXIT>
</CFIF>

<CFIF FindNoCase(".html",Target) EQ 0>
    <HTML>
    <HEAD>
    <TITLE>Error 404</TITLE>
    </HEAD>
    <BODY>
    <!-- Error Message -->
    <H1>ERROR: The requested page could not be found.</H1>
    </BODY>
    </HTML>
</CFELSE>
    <CFSET CFMTarget = Replace(Target, ".html", ".cfm", "ONE")>
    <CFLOCATION URL="#CFMTarget#">
</CFIF>
```

LISTING 4

```
<!-- Find the end of the protocol -->
<CFSET URLStart = (FindNoCase("/", CGI.QUERY_STRING))>
<CFIF URLStart IS NOT 0>
    <!-- Remove information before the URL -->
    <CFSET Target = RemoveChars(CGI.QUERY_STRING, 1, (URLStart -
1))>
</CFELSE>
```

```
<!-- This will force standard processing --->
<CFSET Target="Could not parse">
</CFIF>

<!-- Check for dynamic course info code --->
<CFSET DynamicStart = FindNoCase("DYN_", Target)>
<CFIF DynamicStart IS NOT 0>
    <CFSET DynamicStart = DynamicStart + 4>
    <CFSET DynamicEnd = FindNoCase(".", Target, DynamicStart)>
    <CFSET Target = Mid(Target, DynamicStart, (DynamicEnd - DynamicStart))>
    <CFSET URL.COURSE_ID = "#Target#">
    <CFINCLUDE TEMPLATE="CourseInfo.cfm">
    <CFEXIT>
</CFIF>

<CFIF FindNoCase(".html", Target) EQ 0>
    <HTML>
    <HEAD>
    <TITLE>Error 404</TITLE>
    </HEAD>
    <BODY>
    <!-- Error Message --->
    <H1>ERROR: The requested page could not be found.</H1>
    </BODY>
    </HTML>
<CFELSE>
    <CFSET CFMTarget = Replace(Target, ".html", ".cfm", "ONE")>
    <CFLOCATION URL="#CFMTarget#">
</CFIF>
```

CODE LISTING

The code listing for this article can also be located at

www.ColdFusionJournal.com

Creating a business application with CF and Java servlets

An Online Ticket Store: The Storefront

BY AJIT SAGAR

Sixth in a series of articles focused on using ColdFusion and Java technologies to develop a Ticket Store application. The first was published in the August 1999 issue of *CFDJ* (Vol. 1, issue 4); the next four in *Java Developer's Journal* (Vol. 4, issues 6, 7, 9, 11).

In the August issue of *CFDJ* we walked through the development of a custom CF_Servlet tag that would allow ColdFusion to access Java functionality on the server side via Java servlets. In subsequent issues of *JDJ* I developed the Online Ticket Store application for the Java modules. In order to provide self-contained modules for ColdFusion readers, I'd like to set the context for the design of the modules that have been developed in ColdFusion. Before doing that, however, I want to point out some interesting developments at Allaire that have a direct bearing on this article.

For one thing, the first article in this series generated more feedback than I expected. It has been a pleasant surprise to know that a few companies actually have airline store sites

developed in ColdFusion similar to the fictitious one I've developed through this article series. I've also received e-mail that indicated folks took this application more seriously than I thought they would. These readers concluded that this is a real-world application. While the design of the application outlines a real-world architecture for a business problem and serves as a template for a real-world solution, it's not a production-level system.

Another interesting development was that Allaire acquired Live Software. Most of you are aware of this merger, which happened in June when JavaOne was in progress. While there I spoke with Jeremy Allaire (ColdFusion) and Paul Colton (JRun) about this development – a smart move on Allaire's part in my opinion. A few months before, Allaire had acquired Bright Tiger Technologies, a company that specializes in clustering technology. The acquisition of JRun makes Allaire's app server story complete. ColdFusion is a great product and I've always felt that they needed a story on the Java side. That was part of my motivation for building the Online Ticket Store application. Coincidentally, the August issue of *CFDJ* featured two articles with two individual CF_SERVLET tags – the pro-

fessional one from Live Software and my own.

As the first part of this series was dedicated exclusively to the development of the CF_Servlet tag, I'll begin this article with a description of the application. (Readers who have been following this series in *JDJ* can skip the following section.)

The Online Ticket Store Application

The design of the Online Ticket Store illustrates the use of ColdFusion and the Java platform for building a simple Internet-based Ticket Store application. This application offers online purchase of airline tickets as well as goods sold in airports. One of the objectives of this design is to illustrate how Java servlets can be used as access mechanisms in server modules that serve up data to front-end storefront modules implemented in ColdFusion. The data format for information exchange between the CF page and the Java servlet is XML.

The Online Ticket Store is an Internet-based application that allows an Internet user to log in and purchase tickets via a browser. The store is a front end for the services offered via airline-agency back offices. These back-office locations provide ticket price/availability infor-

mation and accept ticket reservations. The store offers the following services:

- It acts as a “ticket agent,” and may be used to purchase tickets, comparison-shop between different airlines and determine flight itineraries.
- It provides an interface to a virtual airline store that allows catalog sales of merchandise such as clothes, appliances, computer equipment and other goods usually offered in airline magazine catalogs.
- In addition, the store allows Internet users to lease equipment that they can use during the course of the flight. The equipment includes laptops and printers, portable CD players and music CDs. The idea is that a person who uses the Internet to reserve a flight can also instruct the airline to have the leased equipment available on the flight he or she will be taking. The passenger will have the equipment available when he or she boards, and will surrender it on leaving the plane.

The software components that make up the Online Ticket Store, their functions and the technologies used for their implementation are defined in Table 1. We're not too concerned about the airline's back offices. Rather, our focus on the back office is limited to the mechanisms used to access the services offered by the back office. For example, a discussion on how an airline prices and reserves tickets is beyond the scope of this article, as are security issues involved in transporting the data.

Application Framework

The framework for this application is distributed among the following tiers:

- **Client UI:** This is the end-user interface into the Ticket Store. The Client UI is responsible for the front-end interaction with the customer and the connection to the data tier that maintains the Internet user's data and conducts transactions with the Internet user. In our application this is typically a Web browser.
- **Merchant Server tier:** This is where the application server that serves data to the Client UI resides. It has a data store for the customer's profile and maintains the shopping cart, catalog for merchandise, and so forth. It interacts with the Services Access tier to get information from the various data sources (airlines).
- **Services Access tier:** This is a middleware tier that accepts service requests from the Application Server tier, routes them to the Application Services tier and serves back the response to the Merchant Server tier.
- **Application Services tier:** This tier offers the ticket price/availability quote services.

Figure 1 illustrates the application framework tiers.

Component	Function	Technologies
Personal Profile Manager	Maintains personalized profile for end customer. Includes data on customer's flight preferences, frequency of transactions, history of goods purchased, etc. May be updated by customer.	ColdFusion, Microsoft Access
Customer Profile Database	Contains customer profile including name, address, personal preferences, purchase history, purchasing trends, etc. Can be used later to provide CRM (Customer Relationship Management).	Microsoft Access
Catalog	Catalog of goods offered by store, including merchandise available for both purchase and lease.	ColdFusion, Microsoft Access
Payment Manager	Manages payments made by customer while interacting with system. Consists primarily of credit-card-based payment management.	ColdFusion
Login Manager	Manages user login, authentication, passwords.	ColdFusion
Shopping Cart	Keeps track of customer's current purchase as well as pending orders for purchased goods.	ColdFusion
Ticket Reservation and Sales Broker	Main operational module for conducting transactions related to online ticket sales.	ColdFusion and Java Servlets
Merchandise Sales and Leasing Broker	Main operational module for conducting transactions related to online sale and leasing of goods offered in store. Broker accepts orders from customer, runs them against Order Manager, returns confirmation to customer	ColdFusion and Java Servlets
Travel Profile Manager	Keeps track of customer's travel history, preferred airlines, etc. Data captured here and maintained by Ticket Store will be of interest to airlines that collaborate with store.	Java Servlets, Microsoft Access
Order Manager	Interfaces with back-office modules to check availability and to place orders. Placing an order creates a shipment (in the case of PURCHASE) or makes item available at airport (in the case of LEASE).	Java Servlets, RMI
Ticket Pricer	Contains information about products.	Java Servlets, RMI

TABLE 1: Online Ticket Store application modules and related technologies

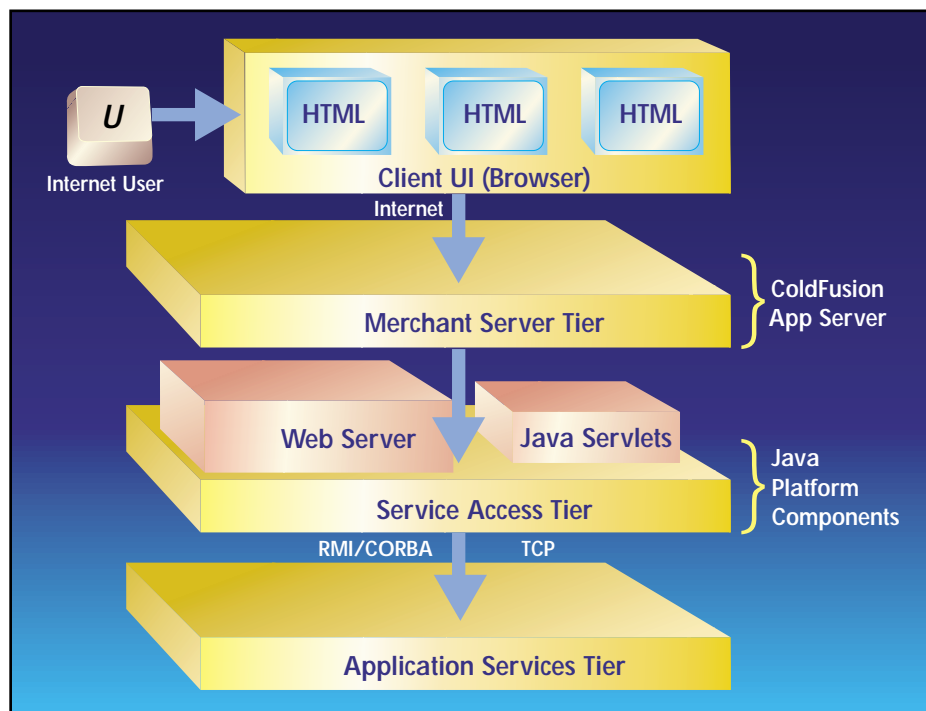


FIGURE 1: Online Ticket Store application framework

rently hard-coded, i.e., they don't get data from an actual database. The purpose of this exercise is to demonstrate how the TicketBrokerServlet and the StoreServlet can be used for ticket reservation and equipment lease operations, respectively. Under the assumption that readers are familiar with XML, I haven't provided any discussion on XML parsing and usage.

The complete Java and CF code is available at www.sys-con.com.

Listing 1: ticket_query.cfm

This page is responsible for inputting the criteria for the flight availability search and passing on the information to the next page, which in turn calls the TicketBrokerServlet. The code in this file is fairly simple. In all of the pages the first code checks to see if the parameter "trace" has been passed in. If it has, and if it's set to "ON," the consequent pages will display debugging output.

```
<CFIF IsDefined("trace")>
  <CFIF trace IS "ON">
    <i><b>Trace is ON</b></i><br><br>
  </cfif>
<CFELSE>
  <CFSET trace="OFF">
</cfif>
```

The code basically consists of a form that accepts user input and passes it on to the next page (invokeTicketBrokerServlet.cfm). Figure 3 shows the screen for this.

Listing 2: invokeTicketBrokerServlet.cfm

The code in this file accomplishes three things:

1. Formats the date passed from the previous page into XML
2. Invokes the TicketBrokerServlet (via the CF_Servlet tag) and passes the XML string to the server for processing
3. Displays the results of the query

Figure 4 shows the entire page for this file. The XML formatting is straightforward. There are parsers available in scripting (e.g., Xparse for JavaScript) that I could have used, but I chose not to complicate the code. The elements of the XML string are created and then concatenated into the "xml_string" variable. Remember that if the server accepted WDDX, we could have serialized the data into WDDX objects and then passed it to the server environment. However, we're working under the premise that the server side accepts XML.

Next, the TicketBrokerServlet is invoked via the CF_Servlet tag. The details of how this works are available in the August *CFDJ*, which is on the **SYS-CON** Web site.

```
<CF_Servlet servletclass=TicketBrokerServlet
serverurl=local host
trace=#trace# method=POST
```

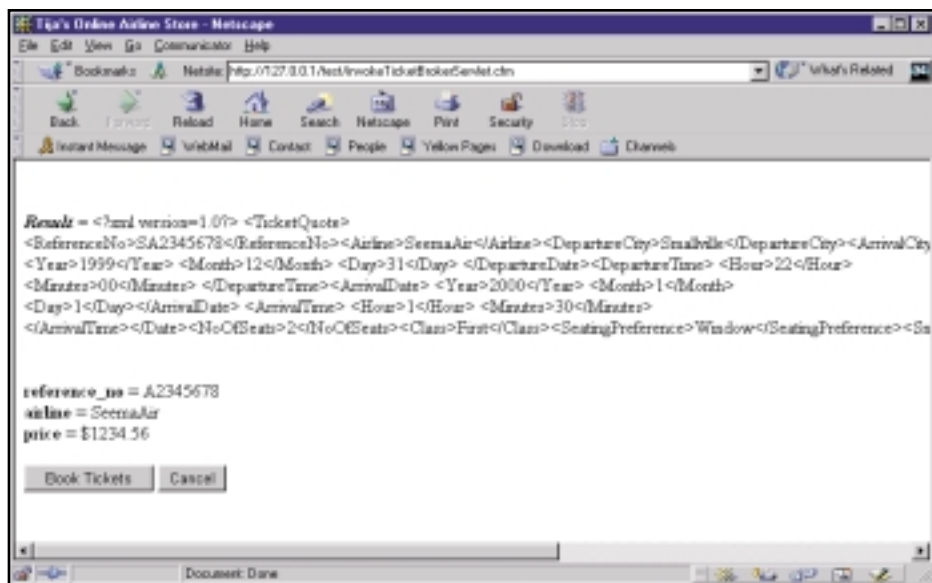


FIGURE 4: Results from TicketBrokerServlet

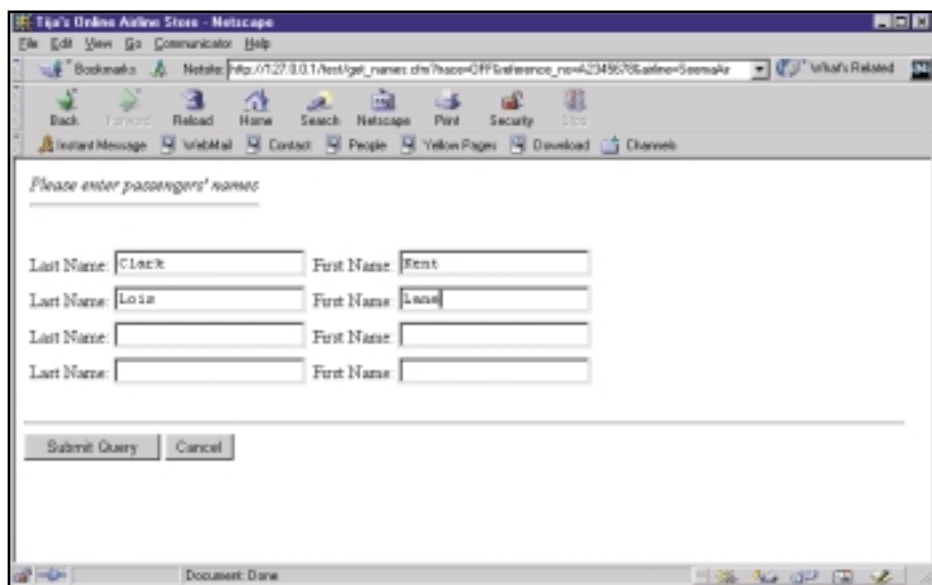


FIGURE 5: Enter passenger names

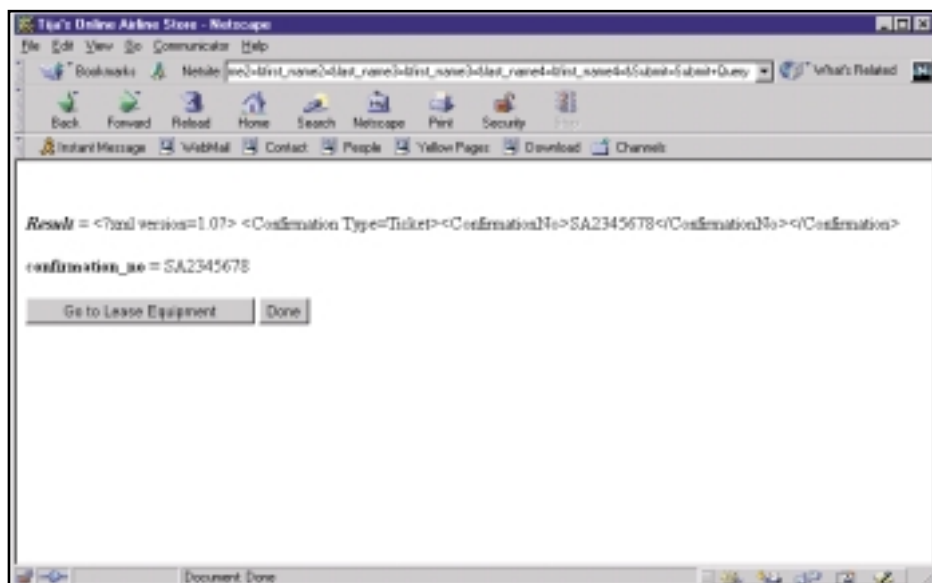


FIGURE 6: Confirmation from TicketBrokerServlet

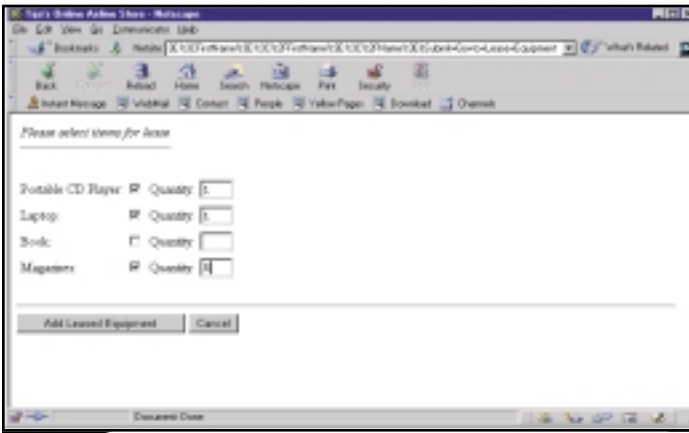


FIGURE 7A Enter Lease equipment request

```
string=#HTMLEdi tFormat(xml_string)#>
```

Last, the result is displayed. `HTMLEditFormat` is used so the reader can view the XML tags also. This output should be parsed and displayed to the user. However, for the purposes of this exercise I have demonstrated only the XML that came back from the servlet.

```
<CFOUTPUT><BR><BR><B><i>Resul t = </i></b>>#HTMLEdi tFormat(resul t)#
```

Three parameters, `ReferenceNo`, `Airline` and `Price`, are passed to the next page and are thus manually parsed from the XML string returned by `TicketBrokerServlet`. `ReferenceNo` is used for all future references to the transaction in this particular use case. At this stage the user can go on to book the ticket or cancel. If the "Book Tickets" button is pressed, the next screen shows up.

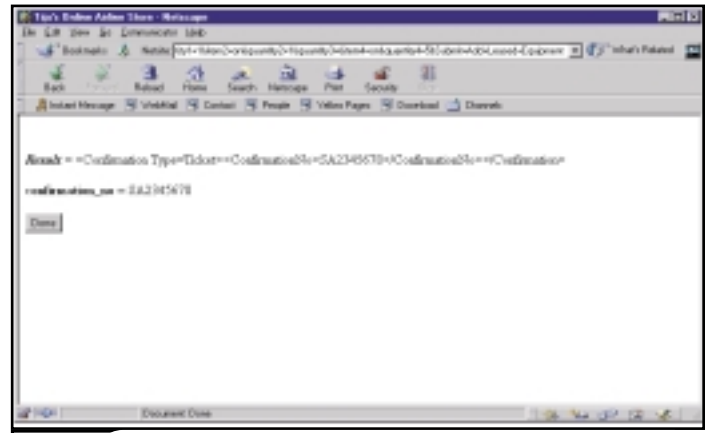


FIGURE 8A Lease Confirmation from StoreServlet

Listing 3: `get_names.cfm`

This is a simple page that accepts the names of the passengers. Up to eight names are allowed. Figure 5 illustrates the form for this page.

Listing 4 : `book_tickets.cfm`

This file is similar in functionality to `invokeTicketBrokerServlet`. It uses the parameters passed from the previous two pages to create a booking request (in the form of an XML string) for the tickets that is passed to the `TicketBrokerServlet`. Note that this is the second invocation of the `TicketBrokerServlet`. The result of this second invocation is parsed and displayed. The result is a simple Confirmation Number. For simplicity I have kept this number the same as the reference number originally returned. Figure 6 illustrates the screen for this page.

At this point the user can select "Done" or go on to lease equipment for the flight by pressing the "Go to Lease Equipment" button. Figures 7 and 8 illustrate the next two screens.

Salient Features

Obviously, this is a very simplified version of a real use case. A lot more database and graphics design will go into actually developing the storefront for this application. However, I hope the code described here serves to illustrate how Java servlets and ColdFusion can be used in conjunction to create a business application. Some of the points:

- There is a continuity in the workflow, illustrated by the two-step process for searching for a flight and then booking the actual tickets. Note that the `TicketBrokerServlet` maintains state for the duration of this transaction.
- Once the custom tag is available (whether my hacked version or Allaire's professional one), it's simple to invoke Java servlets from ColdFusion. This opens up a whole new world – that of Java – for ColdFusion applications. Servlets are Java's gateway to the server side. By using a custom tag, it's simple to get access to server-side functionality.
- Server-side functionality can be distributed across different servlets. In this application CF provides the synchronous user workflow that works across different servlets (`TicketBrokerServlet` and `StoreServlet`).

I hope to develop this application further to bring out other interesting aspects of Allaire's current initiative to integrate with Java APIs.

About the Author

Ajit Sagar is a member of the technical staff at i2 Technologies in Dallas, Texas, focusing on Web-based e-commerce applications and architectures. A Sun-certified Java programmer with nine years of programming experience, including four in Java, Ajit holds an MS in computer science and a BS in electrical engineering.

ajit@sys-con.com

enterprise usa

www.enterpriseusa.com

allaire

www.allaire.com

LISTING 1: ticket_query.cfm

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
<title>Tija's Online Airline Store</title>
<link rel="stylesheet" type="text/css"
href="/automation/style_sheet.css">
</head>

<body>

<CFIF IsDefined("trace")>
<CFIF trace IS "ON">
<i><b>Trace is ON</b></i><br><br>
</cfif>
<CFELSE>
<CFSET trace="OFF">
</cfif>

<!-- General instructions -->

<table class="instructions">

<tr>
<td><i>Enter your flight search criteria:</i><hr></td>
</tr>
</table>

<FORM ACTION="invokeTicketBrokerServlet.cfm" METHOD="POST">
<INPUT TYPE="Hidden" NAME="trace" VALUE=<CFOUTPUT>#trace#</cfoutput>
<table>
<tr width="10"><td class="emphasis">Destination:</td>
<td><select name="departure_city" SIZE="1">
<option VALUE="">-- DEPARTURE CITY --
<option>BludHaven
<option>Central City
<option>Gotham
<option>Metropolis
<option>Smallville
<option>Star City
</select> <td></tr>

<td><select name="arrival_city" SIZE="1">
<option VALUE="">-- ARRIVAL CITY --
<option>BludHaven
<option>Central City
<option>Gotham
<option>Metropolis
<option>Smallville
<option>Star City
</select> <td>

<tr><td class="emphasis">Begin Date:</td>
<td>month: <INPUT TYPE="Text" SIZE="2" NAME="begin_month">
day: <INPUT TYPE="Text" SIZE="2" NAME="begin_day">
year: <INPUT TYPE="Text" SIZE="4" NAME="begin_year"></td></tr>

<tr><td class="emphasis">End Date:</td>
<td>month: <INPUT TYPE="Text" SIZE="2" NAME="end_month">
day: <INPUT TYPE="Text" SIZE="2" NAME="end_day">
year: <INPUT TYPE="Text" SIZE="4" NAME="end_year"></td></tr>

<tr><td class="emphasis">No. of seats:</td>
<td><INPUT TYPE="Text" SIZE="4" NAME="no_of_seats"></td></tr>

<tr><td class="emphasis">Class:</td>
<td><input TYPE="radio" NAME="class" VALUE="Coach"
CHECKED=Coach
<input TYPE="radio" NAME="class" VALUE="First">First</td></tr>

<tr><td class="emphasis">Seating Preference:</td>
<td><input TYPE="radio" NAME="seating_preference"
VALUE="Window" CHECKED=Window
<input TYPE="radio" NAME="seating_preference"
VALUE="Aisle">Aisle</td></tr>

<tr><td class="emphasis">Smoking Preference:</td>
<td><input TYPE="radio" NAME="smoking_preference"
VALUE="NO" CHECKED=Non-smoking
<input TYPE="radio" NAME="smoking_preference"

```

```

VALUE="YES">Smoking</td></tr>

```

```

</table>
<HR>

```

```

<INPUT TYPE="Submit" NAME="Submit" VALUE="submit">
</form>

```

```

</body>
</html>

```

LISTING 2: invokeTicketBrokerServlet.cfm

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
<title>Tija's Online Airline Store</title>
</head>

<body>

<CFIF IsDefined("trace")>
<CFIF trace IS "ON">
<i><b>Trace is ON</b></i><br><br>
</cfif>
<CFSET trace=#trace#>
<CFELSE>
<CFSET trace="OFF">
</cfif>

<!-- Create individual XML elements -->

<CFSET begin_tag="<?xml version=1.0?> <TicketQuery>">
<CFSET departure_city = "<DepartureCity>#form.departure_city#</DepartureCity>">
<CFSET arrival_city = "<ArrivalCity>#form.arrival_city#</ArrivalCity>">
<CFSET begin_date =
"<BeginDate><Year>#form.begin_year#</Year><Month>#form.begin_month#</Month><Day>#form.begin_day#</Day></BeginDate>">
<CFSET end_date =
"<EndDate><Year>#form.end_year#</Year><Month>#form.end_month#</Month><Day>#form.end_day#</Day></EndDate>">
<CFSET no_of_seats = "<NoOfSeats>#form.no_of_seats#</NoOfSeats>">
<CFSET class = "<Class>#form.class#</Class>">
<CFSET seating_preference = "<SeatingPreference>#form.seating_preference#</SeatingPreference>">
<CFSET smoking_preference = "<SmokingPreference>#form.smoking_preference#</SmokingPreference>">
<CFSET end_tag="</TicketQuery>">

<!-- Combine into one string -->
<CFSET xml_string = "#begin_tag# #departure_city# #arrival_city# #begin_date# #end_date# #no_of_seats# #class# #seating_preference# #smoking_preference# #end_tag#">

<CFOUTPUT>
<CFSET result="">
<CFIF trace is "ON">
#HTMLEditFormat(xml_string)#
</cfif>

<CF_Servlet servletclass=TicketBrokerServlet serverurl=localhost
trace=#trace# method=POST
string=#HTMLEditFormat(xml_string)#>

</cfoutput>

<CFOUTPUT><BR><BR><B><I>Result = </I></b>#HTMLEditFormat(result)#

<!-- Extract the Reference No. and the Airline. -->

<CFSET begin_tag_location = #Find("<ReferenceNo>", '#result#')# + 14>
<CFSET end_tag_location = #Find("</ReferenceNo>", '#result#')#>
<CFSET size = #end_tag_location# - #begin_tag_location#>
<CFSET reference_no = #Mid('#result', '#begin_tag_location#', '#size#')#>

<BR><BR><B>reference_no</b> = #reference_no#

<CFSET begin_tag_location = #Find("<Airline>", '#result#')# + 9>
<CFSET end_tag_location = #Find("</Airline>", '#result#')#>
<CFSET size = #end_tag_location# - #begin_tag_location#>

```

career opportunitites


```

<CFSET airline = #Mid('#result#', '#begin_tag_location#', '#size#')#>
<BR><B>airline</b> = #airline#
<CFSET begin_tag_location = #Find("<Price>", '#result#')# + 7>
<CFSET end_tag_location = #Find("</Price>", '#result#')#>
<CFSET size = #end_tag_location# - #begin_tag_location#>
<CFSET price = #Mid('#result#', '#begin_tag_location#', '#size#')#>

<BR><B>price </b>= #price#

</cfoutput>

<FORM ACTION="">
<CFOUTPUT>
  <INPUT TYPE="Button" NAME="Book" VALUE="Book Tickets"
    onclick="location.href='get_names.cfm?trace=#trace#&
      reference_no=#reference_no#&airline=#airline#' ">
  <INPUT TYPE="Button" NAME="Cancel" VALUE="Cancel"

onclick="location.href='ticket_query.cfm?trace=#trace#' ">
</cfoutput>
</form>

</body>
</html>

```

LISTING 3: get_names.cfm

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
  <title>Tija's Online Airline Store</title>
</head>

<body>

<CFIF trace IS "ON">
  <i><b>LALATrace is ON</i></b><BR><BR>
</cfif>

```

```

<table class="instructions">

  <tr>
    <td><i>Please enter passengers' names</i><hr></td>
  </tr>
</table>

<Table>
<FORM action="book_tickets.cfm">
  <CFOUTPUT>
    <INPUT TYPE="Hidden" NAME=trace VALUE="#trace#">
    <INPUT TYPE="Hidden" NAME="reference_no" VALUE="#reference_no#">
    <INPUT TYPE="Hidden" NAME="airline" VALUE="#airline#">
  </cfoutput>
  <TR><TD>Last Name: <INPUT TYPE="Text" NAME="last_name1"></td>
  <TD>First Name: <INPUT TYPE="Text" NAME="first_name1"></td></tr>
  <TR><TD>Last Name: <INPUT TYPE="Text" NAME="last_name2"></td>
  <TD>First Name: <INPUT TYPE="Text" NAME="first_name2"></td></tr>
  <TR><TD>Last Name: <INPUT TYPE="Text" NAME="last_name3"></td>
  <TD>First Name: <INPUT TYPE="Text" NAME="first_name3"></td></tr>
  <TR><TD>Last Name: <INPUT TYPE="Text" NAME="last_name4"></td>
  <TD>First Name: <INPUT TYPE="Text" NAME="first_name4"></td></tr>
</table>

<BR><HR>
  <INPUT TYPE="Submit" NAME="Submit" NAME="Book Tickets">
  <INPUT TYPE="Button" NAME="Cancel" VALUE="Cancel"

```

```

onclick="location.href='ticket_query.cfm?<CFOUTPUT>trace=#trace#</
cfoutput>' ">
</form>

</body>
</html>

```

LISTING 4: book_tickets.cfm

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>

```

career opportunities

Improve Your Code Readability with Fusedoc

Help for writing reusable code

BY
HAL
HELMS

I'm writing this prior to the turn of the new year, so I trust that somehow civilization bungled through and that you're not reading this by the light of your last candle, alert to the strange sounds outside your door that are growing ever nearer...

With the new year we're introducing a monthly column dedicated to all things Fusebox. First, I want to thank all of you who attended the Fusebox session at the ColdFusion Developer's Conference in Boston. Over 300 people attended the session – and some were turned away for lack of room. While I'd like to attribute that to my galvanic oratorical skills, I suspect that it had more to do with how important the quest is for a methodology that works well with ColdFusion.

Why do we need a methodology? Let me reiterate a little of my presentation at the conference and draw an analogy between the state of programming today and that of rifle making 200 years ago. An odd pairing? Well, consider the similarities:

- Skilled craftspeople are needed to create the product.
- Each product is the unique work of its creator.
- There's a shortage of skilled workers.
- The quality of finished goods varies widely.
- Maintenance of our products requires the user to go back to the creator or to employ the skills of a highly paid master craftsperson.
- There's no ability to interchange parts from one product to another.
- Division of labor is very limited.
- The efficiencies of scale are few: making 10 products takes 10 times as long as making one.

Into this highly inefficient situation stepped an American genius: Eli Whitney. Whitney had already achieved great fame (but not wealth) with his cotton gin. Assessing the situation, Whitney decided that he was just the one to solve the "rifle crisis" and pro-

posed to the new national government that he be awarded a contract to produce 10,000 rifles at a cost of just over \$13 each.

The idea was preposterous – but Whitney had already proved his ability to effect miracles. Thus the largest single financial transaction in the young nation's history was agreed upon. Whitney's plan for meeting the contract was to standardize parts so they could be reused among different rifles. The rest, of course, is history. (Perhaps, however, Whitney should be the patron saint of programmers, as he delivered late on the contract.)

Today we face a "software crisis." According to one magazine, two thirds of corporate software projects are never deployed or are declared failures. I think you'll agree those numbers are stunning. But what else could we expect with the situation outlined above?

In other issues of *CFDJ* I've argued that one of the great reasons for Allaire's huge success with ColdFusion is the degree to which it masks complexity from developers, letting them get code out of the programming shop and into use. What we've lacked is a methodology that would direct our programming efforts to produce maintainable, reusable code. I believe that Fusebox goes farther along that path toward radical simplicity in our development efforts.

If we're to get beyond the practice of "one-off" software, in which each bit of code is written one piece at a time, we need to agree on certain standards. We need to document our code with the understanding that it will be fitted into other applications and maintained by different coders many times during its life. And, as we've just recently seen with the Y2K issue, there's no telling

just how long that life may be.

This is nothing new, of course; we know documentation is good practice. Why, then, is it so rare? Perhaps we feel something like Mark Twain did when he was asked by a reporter whether there was anything he feared. Twain mulled the question over before drawing, "A blank piece of paper." It's intimidating to think we have to engage in writing a novella with every code file. What exactly should we put in it? What format should it have?

You may already have a standard for documentation. If not, I suggest one for you to evaluate. I call this *Fusedoc* – an appropriate name as the idea was borrowed from its cousin, Java, with its Javadoc specification. Fusedoc was written to work with Fusebox, but it can be used with any code files. It's usually the first step I take when helping a new company implement Fusebox as it's easy to adopt and provides immediate benefits.

The Fusedoc format I took out of one of my fuses is given in Listing 1. One of these goes at the top of every code file.

Here's how to read the Fusedoc:

```
<!-- actValidateUser.cfm -->
```

The name of the fuse is set in an HTML comment, so if we do a "View Source" from within a browser, we can immediately tell what files are in play.

```
<!--
|| I make sure that the username and
password given to me are valid
entries in the USERS table. If they
are, I will return to...
```

This is the "Responsibilities" section of the Fusedoc. I write this in the

allaire

www.allaire.com

**Take a look
at our specials
this month!**

EASTLAND DATA SYTEMS Internet Shopping with Java Shopping Cart

... Described as the most progressive and interactive form of shopping on the web today... This Java Applet provides a complete user interface package for Internet Shopping Web Sites. Using Java technology we produce a drag-and-drop shopping user interface that is fun and easy to use, encouraging shoppers instead of frustrating them with confusing controls that are hard to follow. And the easier it is to shop, the more you sell.



\$294⁹⁹

Hybrid Shopping Cart

This Java Applet provides a complete user interface package for Internet Shopping Web Sites. A "Hybrid" is defined as an offspring of two varieties. A blending of the best features from our CGI and Java shopping products, we took the most powerful aspects of Java technology: real-time, on-screen updating and computational capabilities. And combined those with the most desirable features of our CGI shopping Cart, namely it's flexibility and compatibility with web designers with artistic talent.



\$294⁹⁹

CGI Shopping Cart

The Shopping Cart automates the Shopping Process to make shopping on your site intuitive, straight forward, and enjoyable! It's one of the most affordable Shopping Carts because it was designed for small businesses. Specifically for entrepreneurs who are testing the Internet waters, and can't or don't want to make large investments into bells and whistles for their site. But simply want to make shopping on their site easy for the customer.



\$294⁹⁹

Guaranteed Best Prices

JDJ Store Guarantees the Best Prices. If you see any of our products listed anywhere at a lower price, we'll match that price and still bring you the same quality service.

Terms of offer:

- Offer good through October 31, 1999
- Only applicable to pricing on current versions of software
- August issue prices only
- Offer does not apply towards errors in competitors' printed prices
- Subject to same terms and conditions

Prices subject to change.
Not responsible for typographical errors.

Attention Java Vendors:
To include your product in JDJStore.com,
please contact jackie@sys-con.com



**GUARANTEED
BEST PRICES
FOR ALL YOUR
JAVA RELATED
SOFTWARE
NEEDS**

ALLAIRE

ColdFusion 4.0

With ColdFusion 4.0, create Web applications for self-service HR solutions, online stores, interactive publishing and much more. It's the integrated development environment that has all the visual tools you need to create Web applications quickly and easily. From simple to sophisticated ColdFusion gives you the power to deliver the Web solutions you need - faster, and at a lower cost.



ColdFusion Studio 4.0 **\$354⁹⁹**
SkillBuilding with ColdFusion Interactive Training CD **\$284⁹⁹**

ALLAIRE

JRun - Live Software

JRun is the industry-leading tool for deploying server-side Java. JRun is an easy-to-use Web server "plugin" that allows you to deploy Java Servlets and JavaServer Pages. Servlets form the foundation for sophisticated server-side application development. Java servlets are platform independent, easy to develop, fast to deploy, and cost-effective to maintain.



JRun **\$558⁹⁹**

PROTOVIEW

Java Enterprise Editions

The Java Enterprise Editions offer you a choice between comprehensive packages of award-winning AWT or JFC components along with enterprise-level support and subscription service. Powerful, extendable, lightweight components built on the foundation of JFC, the ProtoView JFCSuite contains JFCDataCalendar, JFCDataExplorer and JFCDataInput. The JSuite (AWT) includes the DataTableJ grid component with JDBC and Visual Café database support. Also includes TreeViewJ, CalendarJ, TabJ and WinJ.

JSuite Enterprise Edition **\$818⁹⁹**
JFC Enterprise Edition **\$818⁹⁵**
JSuite **\$328⁹⁹**
JFCSuite **\$408⁹⁹**

GALILEO DEVELOPMENT SYSTEMS

Intr@Vision Foundation

Intr@Vision Foundation helps bring ColdFusion development to the next level. It provides an out-of-the-box application security architecture for handling your most complex intranet and extranet needs. Instead of spending 30% of your development time adding security to every application you build, it gives you a proven solution with a single line of code. Intr@Vision Foundation allows your developers to focus on building business solutions, not infrastructure.



Intr@Vision Foundation **\$3499⁹⁹**

ALLAIRE

HomeSite 4.0

HomeSite is the award-winning HTML editing tool that lets you build great Web sites in less time, while maintaining Pure HTML. Unlike WYSIWYG authoring tools, HomeSite gives you precise layout control, total design flexibility and full access to the latest Web technologies, such as DHTML, SMIL, Cascading Style Sheets and JavaScript. HomeSite 4.0 is the only HTML editor featuring a visual development environment that preserves code integrity.



HomeSite 4.0 **\$87⁹⁹**

INSTALLSHIELD

InstallShield Java Edition 2.5

InstallShield Java Edition 2.5 is the powerful tool developers require to produce bulletproof InstallShield installations with Java versatility. You can target your application for multiple systems with cross-platform distribution. And InstallShield Java Edition 2.5 offers the key features and functionality designed to let developers go further in distribution and deployment.



InstallShield Java Edition **\$474⁹⁹**

KL GROUP

JProbe Suite

JProbe Profiler is the most powerful tool available for finding and eliminating performance bottlenecks in your Java code. JProbe Coverage makes it easy to locate individual lines of untested code and reports exactly how much of your Java code has been tested. JProbe Threadalyzer lets you pinpoint the cause of stalls and deadlocks in your Java applications and makes it easy to predict race conditions that can corrupt application data.



JProbe Profiler w/ Standard Support (inc. JProbe Memory Debugger) **\$464⁹⁹**
JProbe Coverage w/ Standard Support **\$464⁹⁹**
JProbe Threadalyzer w/ Standard Support **\$464⁹⁹**
JProbe Suite w/ Standard Support **\$934⁹⁹**

CATOUZER

Synergy SOHO

Synergy is a ColdFusion-based Web application framework for instantly deployable corporate intranets. It consists of an Application Services Layer, which offers central security and administrative services, and eight core collaborative applications. Synergy's open architecture is designed for implementing existing ColdFusion applications and developing new ColdFusion applications specifically tailored to the customer's needs.



Synergy SOHO **\$499⁹⁹**

ORDER ONLINE

WWW.JDJSTORE.COM

first person (as if the fuse were speaking) and try to provide enough information for another person to tell immediately what's happening with the fuse. My goal is to provide enough for a CF coder, armed with a copy of the data schema, to write this fuse without any knowledge of the application in which it'll be used. The double pipe symbols (||) will be used by our "magic box" to extract individual pieces of the Fusedoc.

|| hal.helms@TeamAllaire.com

You may want to keep historical information here about revisions, testing, etc.

```
||
--> userName: a STRING
--> password: a STRING
--> RFA.successfulValidation: a valid FUSEACTION
--> RFA.failedValidation: a valid FUSEACTION
<-- [badLogin]: "yes"
<--
++> dsn: an APPLICATION variable ODBC datasource
+++
```

These are the various arguments being passed into and out of the fuse. There is a key to the meaning of the prefixes:

```
--> an incoming parameter: the text following the prefix provides more information about the parameter
--> [square brackets indicate that this incoming variable is optional]
<-- an outgoing parameter
<-- [same explanation as the square
```

```
C:\inetpub\wwwroot\wireframes\actCheckForNoPages.cfm

Responsibilities: I have to make sure that we have the proper pages and configuration.
The possible states are: 1. I was given a pageID, and it does exist; I do nothing. 2. I was
not given a pageID: I need to see if any home page exists. If it doesn't I need to see if
any
pages exist for this application. If the home page does exist, I'm going to set an attrib-
utes variable called pageID with the pageID of the home page. If not, then I need to get the
user to specify a home page for one of the existing pages (RFA.assignHomePage). If not, I
will just have the user create a page (RFA.addPage). 3. I was given a pageID, but it
doesn't exist. I need to treat this as if I was not given any pageID.
Author: hal.helms@teamAllaire.com

Attributes:
--> RFA.assignHomePage: a FUSEACTION if the user needs to assign an existing page as
an isHomePage.
--> RFA.addPage: a FUSEACTION if the user needs to add a new application page
--> [pageID]: a valid PK from the PAGES table
<--
++> applicationID: a CLIENT PK from APPLICATIONS table indicating the current
application.
+++

C:\inetpub\www.root\wireframes\actDeletePage.cfm
Responsibilities: I delete a page from an application. When returning to the fusebox, it
needs to refresh to top.location.href.
Author: hal.helms@teamAllaire.com
```

FIGURE 1: Output from the Fuse Documentor

brackets above]

```
<> a parameter that will be received
by and sent from the fuse without
being changed
++> a global variable not explicitly
passed into the fuse, such as serv-
er, application and session vari-
ables
+++ any file that is required by this
fuse. This may include files,
fusefunctions, etc.
```

|| FUSEDOC----

This just ends the Fusedoc section. I've found this system to be very workable; once used to it, programmers find it very helpful as it concisely communicates a good deal of information. Even pointy-haired managers and customers can look at the code and have a sense of what's going on.

We can make it even more useful if we process this code through our "magic box" - a code file (fuseDocu-mentor.cfm) that parses the individual

elements of the Fusedocs and outputs them on the screen. Fuse Documentor was written to be called as a custom tag. For parameters, it takes...well, let's just run the Fuse Documentor through itself to find out more about it (see Figure 1).

If you want to see particulars on every file in a directory, you can write a little code that will loop through the directory and send each file to Fuse Documentor (see Listing 2). This will let you see very quickly what's going on in an entire code repository.

Fusedoc is a small, lightweight system for seeing the structure of code. I've found it to be helpful in writing maintainable, reusable code. I hope you find it similarly helpful.

You can download the Fuse Documentor file as well as a CF Studio template at www.TeamAllaire.com.

ABOUT THE AUTHOR

Hal Helms is a Team Allaire member living in Atlanta, Georgia. A frequent writer on ColdFusion and Fusebox, he also offers training and mentoring on these subjects.



HELMS@TEAMALLAIRE.COM

LISTING 1

```
<!-- actValidateUser.cfm -->

<!--
|| I make sure that the username and password given to me
are valid entries in the USERS table. If they are, I'll
return to the fusebox with an RFA (return fuseaction) of
successfulValidation. If not, I'll return to the fusebox
with an RFA of failedValidation. If the user fails, I'll
also pass back a variable called "badLogin" set to "yes"

|| hal.helms@TeamAllaire.com
||
--> userName: a STRING
--> password: a STRING
--> RFA.successfulValidation: a valid FUSEACTION
--> RFA.failedValidation: a valid FUSEACTION
<-- [badLogin]: "yes"
<--
++> dsn: an APPLICATION variable ODBC datasource
+++
```

|| FUSEDOC----

LISTING 2

```
<cfoutput>

<cfdirectory
sort="name ASC"
action="LIST"
name="dirList"
directory="#GetDirectoryFromPath(GetTemplatePath())#">
</cfoutput>

<cfoutput query="dirList">
<cf_fuseDocumentor
fuseName = "#GetDirectoryFromPath(GetTemplatePath())#name#">
</cfoutput>
```

ADVERTISING INDEX

ADVERTISER	URL	PH	PG
ABLE SOLUTIONS	WWW.ABLECOMMERCE.COM	360.253.4142	2
ALLAIRE	WWW.ALLAIRE.COM	888.939.2545	21,31,43,49
BIZNIZ WEB	WWW.WEBPUBLISHINGTOOLST.COM	281.367.4016	52
CAREER OPPORTUNITIES			45,46
CATOUZER	WWW.CATOUZER.COM	604.662.7551	55
CFDEV.COM	WWW.CFDEV.COM	800.854.7838	47
COMPUTERJOBS.COM	WWW.COMPUTERJOBS.COM		3
CONCEPTWARE AG	WWW.CONCEPTWARE.COM	061.964.7320	4
DATARETURN	WWW.DATARETURN.COM	800.767.1514	9
DIGITAL NATION	WWW.DEDICATEDSERVER.COM	703.642.2800	35
EKTRON	WWW.EKTRON.COM	603.594.0249	13
ENTERPRISE USA	WWW.ENTERPRISEUSA.COM	248.888.1473	42
EPRISE	WWW.EPRISE.COM	800.274.2814	11
FUNDERE SOFTWARE	WWW.FUNDERE.COM	310.392.1005	15
INFOBOARD	WWW.INFOBOARD.COM	800.514.2297	47,53
INFORUM SOLUTIONS	WWW.INFORUMSOLUTIONS.COM	978.887.0080	17
INTERMEDIA	WWW.INTERMEDIA.NET	650.424.9935	56
NEO TECH	WWW.CF-TRAINING.COM	973.540.9672	47
ON-LINE DATA SOLUTIONS	WWW.COOLFUSION.COM	516.737.4668	54
ORIGENT	WWW.ORIGENT.COM	617.623.8044	19
PIRANHANET	WWW.PIRANHANET.COM	206.374.0880	37
RSW SOFTWARE	WWW.RSWSOFTWARE.COM	508.435.8000	23
SD 2000	WWW.SDEXPO.COM		29
SITEHOSTING.NET	WWW.SITEHOSTING.NET	888.463.6168	27
VIRTUALSCAPE	WWW.VIRTUALSCAPE.COM	212.460.8406	34
WORLDWIDE NETFAST	WWW.NETFAST.NET	800.362.9004	54

Get Your Own Subscriptions to the Finest Technical Journals in the Industry!

1-800-513-7111
www.sys-con.com

Able Solutions

Enter the realm of browsable store building and administration - from your browser. Build "your_site.com" with secure Merchant Credit Card Processing. Maintain inventory, add discounts and specials to keep your customers coming back. Increase sales with cross selling and membership pricing.

11700 NE 95th Street, Suite 100, Vancouver, WA
www.ablecommerce.com • 360 253-4142

Catouzer, Inc.

With the Synergy 1.0 Web application framework, creating custom intranet applications is a breeze. The Synergy Application Development Kit (ADK) gives you the tools to rapidly develop your custom applications, which can be fully integrated and managed under the Application Services Layer (ASL).

1228 Hamilton Street, Suite 501, Vancouver, B.C. V6B 2S8, Canada
www.catouzer.com • 604 662-7551

CFDEV.com

CFDEV.COM is fine-tuning Site-a-Matic, its soon-to-be-released Web Developer's Toolkit. Site-a-Matic is an application management program that allows developers to quickly deploy applications, reducing development time from hours to minutes. In one week CFDEV.COM received over 300 downloads and plans to add new custom tags each week. Free tags are available at the site as well.

www.cfdev.com 1-800-791-1916

Computerjobs.com

ComputerJobs.com is a leading Internet-based job search company dedicated to helping computer and information technology (IT) professionals find great jobs. Our user-friendly, proprietary Web site provides IT job seekers and hiring companies a convenient, effective way to connect. In addition to thousands of job listings for candidates, we provide companies who list jobs with us access to résumés of the hottest IT professionals on the Web.

3200 Windy Hill Road, Suite 700 West, Atlanta, GA 30339
www.computerjobs.com

Conceptware

Conceptware provides professional solutions and first-class products for the I*Net. With the launch of Spectra, the new Content Management and e-commerce framework by Allaire Corporation, Conceptware has announced its Internet suite, GateBuilder, which is based on Spectra.

Industriestraße 30-34, Eschborn, Germany 65760
www.conceptware.com +49-(0)6196-47 32-0

Data Return Corporation

Data Return offers extensive support for customers utilizing ColdFusion from Allaire. With customers delivering over 50,000 user sessions per day, we know how to provide high availability solutions for this advanced application server. Our technical support staff has extensive experience in coding custom ColdFusion Tags as well as managing Microsoft SQL server. We also support CyberCash for customers interested in real-time credit card processing. Our combination of support and experience offer an ideal environment for deploying your applications developed for ColdFusion.

801 Stadium Dr., Ste 117, Arlington, TX 76011
www.datareturn.com • 800 767-1514

digitalNATION - a VERIO company

digitalNATION, VERIO's Advanced Hosting Division, is the world's leading provider of dedicated server hosting, with over 1,650 servers online today. dN's superior connected network and service abilities have earned dN a solid reputation as a first-choice provider of dedicated server solutions (Sun, Windows NT, Linux and Cobalt). digitalNATION has been providing online and network services for over six years. One of the first ISPs to provide dedicated servers running Microsoft Windows NT, the dN staff has unparalleled experience in this industry.

5515 Cherokee Ave, Alexandria, VA 22312-2309
www.dedicatedserver.com • 703 642-2800

To place an ad in the ColdFusion Marketplace contact Robyn Forma at 914 735-0300

Ektron

Ektron supports the next-generation needs of e-businesses by providing dynamic Web infrastructure solution tools designed for use by nontechnical staff. Ektron's flagship offering, eContentManager, gives staff members across an organization the hands-on ability to make real-time additions and updates to Web content without requiring knowledge of a programming language -- while still allowing for centralized administrative control and security. With competitive advantages such as ease-of-integration and drag & drop everything, Ektron is looking to provide these empowering products to customers, resellers and integrators.

5 Northern Blvd., Suite 6, Amherst, NH 03031
www.ektron.com • 603-594-0249

Enterprise USA, Inc.

Enterprise USA is a software product and computer consulting firm based in the Metropolitan Detroit area of Michigan, with offices in Farmington and Lansing. Enterprise USA specializes in providing computer consultants for Internet development, Linux and the Microsoft product line. In particular they provide supplemental staffing of computer programmers, help desk professionals, network integration specialists, and project managers.

33425 Grand River Ave, Suite 201, Farmington, MI 48335
www.enterpriseUSA.com • 248 888-1473

Eprise Corporation

If your customers are looking for a content management solution, Eprise Participant Server FastStart Kit for Allaire ColdFusion developers can save you time and resources. Participant Server is a flexible-content management framework that enhances high-value business relationships through the delivery of timely, targeted, Web-based communications.

1671 Worcester Road, Framingham, MA 01701
www.eprise.com 800 274-2814

Fundere Software

Fundere Corporation, a leading ColdFusion developer, offers software solutions that enable customer service organizations to respond, track and analyze high volumes of customer inquiries. Fundere's SupportAgent software has a built-in natural language engine that allows customers to enter their queries using plain English. Built on top of ColdFusion, the SupportAgent can leverage any existing Web application or legacy database.

18 Horizon Avenue, Venice, CA 90291
www.fundere.com 310 392-7805

Highlight your website with
infoboard
NT and UNIX
Cold Fusion Hosting
Development Consulting
Oracle, Informix, MS SQL, E-Commerce Plug-ins
1-800-514-2297
sales@infoboard.com
www.infoboard.com

INFORUM Solutions

INFORUM Virtual Office Park integrates communication, data collection, knowledge management, office management, and planning tools in a single Web-based system. With it any nontechnical individual or organization can create custom, interactive, fully functional Web sites in a matter of hours, not weeks! Link virtual offices to form virtual office parks. You and your most far-flung colleagues can collaborate more effectively, manage projects more successfully, and achieve goals more quickly.

458 Boston Street, Topsfield, MA 01983
www.inforumsolutions.com 800 442-0599

Intermedia, Inc.

Our advanced virtual hosting packages (powered by Microsoft Windows NT and Internet Information Server 4.0) offer an environment supporting everything today's advanced Web developer or sophisticated client could ask for. Complete ODBC support is available on plans B and C. We support Microsoft Index Server on all hosting plans.

953 Industrial Avenue, Suite 121, Palo Alto, CA 94303
www.intermedia.net • 650 424-9935

NetFast

NetFast strives to become the foremost leader in Web application software. Our applications empower and enable function-building efforts of Web developers and designers. These ready-to-use software components can be bundled

into existing Intranet and Internet sites to increase usability, drive traffic and differentiate online offerings. The fusion of application development and hosting provides substantial revenue opportunities for VARs and integrators of NetFast products.

6699 Port West Drive, Suite 130, Houston, TX 77024
www.netfast.net 800 362-9004

Get Trained. Get A Job.

JOBSEEKERS:
Search through hundreds of listing from dozens of companies & recruiters at:
www.coldfusionjobs.NET

COLD FUSION TRAINING:
Search for a facility near you, view course schedules, or purchase books & materials.
www.cf-training.com

Looking for Cold Fusion Developers?

ADD US TO YOUR BOOKMARKED SITES!
We offer unlimited job listings and many other services. Mention promotion code 1102X

On-Line Data Solutions

CoolFusion.com is dedicated to providing unique and powerful add-on solutions for ColdFusion development and implementation. The site is hosted and maintained by On-Line Data Solutions, Inc. -- a leader in ColdFusion integration. Our ColdFusion integration products line is called inFusion -- a combination of "infuse" and ColdFusion. For information about our flagship product, inFusion Mail Server, we invite you to read the online information (where you can also download the latest beta) and join the inFusion Mail Server support list.

24 Elm Street, Centereach, NY 11720-1706
www.coolfusion.com 516 737-4668

Origent

Origent Creator for ColdFusion is a powerful RAD tool designed to suit the needs of the ColdFusion professional. Creator will instantly generate a complete, web-based interface to your database from a simple, SQL-like description of your database schema -- Creator will even write the description for you from your ODBC datasource. With Creator, you can create a working, ColdFusion-based web interface to an existing database in under two minutes. In under half an hour, you can create a comprehensive interface that takes into account complex table relationships, field constraints, and other key criteria.

8 Faneuil Hall Marketplace, 3rd Floor, Boston, MA 02109
(617) 973-5136 • www.origent.com

PiranhaNet

PiranhaNet offers complete Web site and information design services. From full-featured Web presences and Web-based document management systems to streaming video and e-commerce, PiranhaNet can deliver a total solution. Our work speaks for itself. Tour a virtual gallery of our featured projects, and learn how PiranhaNet helps clients succeed at www.piranhanet.com/services.htm.

2112 6th Avenue, Seattle, WA 98121
www.piranhanet.com 206 374-0880

RSW Software

RSW Software is a wholly owned subsidiary of Teradyne, Inc., and specializes in Web application testing software. Established with the goal of providing best-in-class testing products, RSW offers a suite of products called the e-TEST Suite, which automates the process of testing business-critical Internet and intranet applications.

44 Spring Street, Second Floor, Watertown, MA 02172
www.rswsoftware.com • 508 435-8000

Sitehosting.NET

Successful electronic commerce starts at SiteHosting.net; a division of Dynatek Infoworld, Inc., which provides total Web development services. We offer personal and efficient customer service with reliability at value prices. All our plans include access to SSL (Secure Socket Layer). We support ColdFusion, Active Server Pages, Real Audio/Video, Netshow Server, and more. Our hosting price starts at \$14.95/month.

13200 Crossroads Parkway North, Suite 360, City of Industry, CA 91746
www.sitehosting.net • 877 684-6784

Virtualscape

Why host with Virtualscape? Nobody else on the Internet understands what it takes to host ColdFusion like we do. Virtualscape is the leader in advanced Web site hosting. From Fortune 500 extranets to e-commerce sites and more, developers recognize our speed, stability, reliability and technical support.

215 Park Avenue South, Suite 1905, New York, NY 10003
www.virtualscape.com • 212 460-8406

Visual Mining Selects Allaire's ColdFusion Server

(Rockville, MD, and Cambridge, MA) – Visual Mining, Inc., a leading provider of enterprise corporate decision portals, and Allaire Corporation announced that Visual Mining has selected Allaire's ColdFusion server to power Visual Mining's corporate portal product, Decision.Control. www.visualmining.com and www.allaire.com.

Ektron Realigns Pricing

(Amherst, NH) – Ektron, Inc., has reduced pricing for its eWebEdit-Pro, eContentManager and eWeb-Builder products, making them available to organizations seeking next-generation Web development and content-management solutions. For more information go to www.ektron.com.

Allaire Announces ColdFusion 4.5 for Linux

(Boston, MA) – Allaire Corporation released a new version of its cross-platform Web application server, ColdFusion. As one of the first commercial application servers for Linux, ColdFusion 4.5 gives the rapidly growing Linux community a powerful new platform for building and delivering enterprise and e-commerce systems. www.allaire.com.

Macromedia Releases Authorware 5.1

(San Francisco, CA) – Macromedia, Inc., introduces Macromedia Authorware 5.1, an upgrade to its authoring software for deploying learning applications for the Web, networks and CD-ROM. The upgrade is free to registered users of Macromedia Authorware 5. For

further information go to www.macromedia.com.

Allaire JRun to Power In-Store Information Kiosks for Home Depot

(Cambridge, MA) – Home Depot, the world's largest home improvement retailer, has chosen Allaire JRun to power its in-store information kiosks. "We were looking for a solution that would enable us to deliver product and vendor infor-

mation quickly and efficiently to our customers via our in-store information kiosks." said Ron Griffin, CIO of Home Depot. For more information on JRun go to www.allaire.com.

WebTrends Launches Enterprise Reporting Server Version 2.0

(Portland, OR) – WebTrends Corporation has released Enterprise Reporting Server 2.0. This version provides customers with advanced reporting features to help turn e-business intelligence into results. With the release of an NT edition, Enterprise Reporting

Server is a true cross-platform Web traffic analysis solution operating on Linux, Solaris and Microsoft Windows NT platforms. For more information or to download a free trial version see www.webtrends.com/products/ers.

worldwide netfast

www.netfast.net

Catouzer

www.catouzer.com

NT WEB HOSTING Intermedia. NET

TAKE
control
Web of your
Site

TEST DRIVE FREE DEMO ACCOUNT



Unlimited Email accounts
Unlimited traffic/bandwidth

New! Clockwork Mail[™]
Manage email from your browser

FOR
FREE
SET UP
USE PROMO CODE
CFDJ

NT Hosting with IIS 4.0
MS SQL Server 7.0, Access, FoxPro
Active Server Pages
Front Page 98 and 2000
Cold Fusion 4.01
SSL Secure Server
Shopping Carts: StoreFront, Drumbeat
ECommerce
CyberCash, PaymentNet
WebTrends Statistics
Reseller Program Available

Instant Control!

Add domains
Add disk space
Add email accounts
Add mailing lists
Register Cold Fusion Custom Tags
Register ODBC
Register Active X DLL and OCX

sign up online
and activate your account in 10 minutes

www.intermedia.net

800.379.7729



Intermedia.NET, Inc.

953 Industrial Ave
Palo Alto CA 94303

sales@intermedia.net