

COLD FUSION Developer's Journal

ColdFusionJournal.com

January/February, 1999 Volume: 1 Issue: 1

FREE access to
Digital Edition p.34

From the Editor
Welcome to CFDJ!

Chad Sittler page 5

Programming Techniques

**Dynamic Variable
Length Forms**

by Geoff Caras page 38

Tips & Techniques

**File Upload Utility
with ColdFusion**

by Matt Newberry page 40

Product Review

**AbleCommerce
Developer 2.6**

by Tom Taulli page 42

Custom Tags:

**Building
Custom Tags
Using ColdFusion**

by Ashley King page 47

CF News page 36

User Groups page 50

RETAILERS PLEASE DISPLAY
UNTIL MARCH 31, 1999

SYS.CON
PUBLICATIONS



Leveraging CF: Putting Client/Server on the Web

10

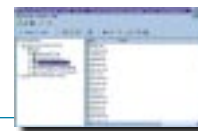
Replicating the User Interface and Functionality

Jerry Bradenbaugh

CFDJ Feature: What's Different About ColdFusion's 4.0

18

**Some of the Enhancements That Make
the Upgrade Worthwhile for Developers**



Richard Schulze

The Data Path: From Database to Web Page and Back

22

Using SQL, HTML and JavaScript to Retrieve Data

Andy Birchall

A Look Inside: Creating a 'Live' ColdFusion Site

26

Requesting Information from the Server

Raymond Camden

ColdFusion & CGI: Transitioning to ColdFusion

30

**ColdFusion and the Web
Go Hand in Hand**



Jim Esten

ColdFusion Slick Tricks: Taming the Cookie Monster

44

Hysteria About Cookies is Wildly Distorted



Bob Siegel

Able Solutions

www.ablecommerce.com

Allaire 1

www.allaire.com

Catouzer

www.catouzer.com

Geoff Caras, Jim Esten, Ben Forta,
Steve Nelson, Richard Schulze

Editor in Chief: Chad Sittler
Art Director: Jim Morgan
Executive Editor: Scott Davison
Managing Editor: Hollis K. Osher
Senior Editor: M'lou Pinkham
Production Editor: Brian Christensen
Product Review Editor: Tom Taulli
Tips & Techniques Editor: Matt Newberry

WRITERS IN THIS ISSUE

Andy Birchall, Jerry Bradenbaugh,
Raymond Camden, Geoff Caras, Jim Esten,
Ben Forta, Ashley King, Matt Newberry,
Richard Schulze, Bob Siegel, Tom Taulli

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: \$49.99/yr. (6 issues)

Canada/Mexico: \$69.99/yr.

Overseas: \$99.99/yr.

Back Issues: \$12 each

Publisher, President and CEO: Fuat A. Kircaali
Vice President, Production: Jim Morgan
Vice President, Marketing: Carmen Gonzalez
Advertising Manager: Robin Forma
Advertising Assistant: Jaclyn Redmond
Accounting: Ignacio Arellano
Graphic Designers: Robin Groves
Alex Botero
WebMaster: Robert Diamond
Customer Service: Sian O'Gorman
Paula Horowitz
Online Customer Service: Mitchell Low

EDITORIAL OFFICES

SYS-CON Publications, Inc.

39 E. Central Ave., Pearl River, NY 10965
Telephone: 914 735-1900 Fax: 914 735-3922
Subscribe@SYS-CON.com

COLD FUSION DEVELOPER'S JOURNAL

is published bimonthly (6 times a year)

for \$49.99 by SYS-CON Publications, Inc.,

39 E. Central Ave., Pearl River, NY 10965-2306.

Application to mail at Periodicals Postage rates is pending
at Pearl River, NY 10965 and additional mailing offices.

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 © 1999 by SYS-CON Publications, Inc. All rights reserved.
No part of this publication may be reproduced or transmitted in any form
or by any means, electronic or mechanical, including photocopy or any
information storage and retrieval system, without written permission.

For promotional reprints, contact reprint coordinator.

SYS-CON Publications, Inc., reserves the right to revise, republish and
authorize its readers to use the articles submitted for publication.

ISSN # 1087-6944

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.

SYS-CON
PUBLICATIONS

Welcome to CFDJ!



If you've picked up the premier issue of *ColdFusion Developer's Journal* expecting to read about a way to create an infinite energy source, there's no need to read further. However, if you've picked up this issue expecting to read about an energy source that can take raw data and turn it into something useful and even pleasing to the eye, then read on.

In 1995 the Allaire Corporation was founded and began its foray into the relatively uncharted territory of electronic commerce and business information systems. The evolution of the Web forced developers to create dynamic, interactive Web sites, but the relative infancy of HTML restricted their ability to do this. A better tool was needed. ColdFusion was the result. Now *ColdFusion Developer's Journal* breaks onto the scene.

CFDJ has amassed articles from the front-runners in the ColdFusion development field, including Ben Forta, the well-known CF evangelist and author of *The ColdFusion Web Database Construction Kit*.

These articles aren't geared solely to experienced CF developers, but to the beginner as well. It's *CFDJ*'s goal to make sure that no issue is beyond the ability of beginners, or too simplistic for those who've been playing with CF tags for years.

The most recent development in the world of CF is ColdFusion 4.0. Newly enhanced properties of previous CF versions – as well as new tags and the ability to create more powerful custom tags – highlight this newest release from Allaire. The ability to integrate other programming languages also makes ColdFusion the database integration tool of both the present and the future.

Without a doubt, the e-commerce industry is becoming one of the most explosive aspects of the Web. Who'd have thought, as they bought their state-of-the-art 2400-baud modem, that they'd one day be buying, selling and trading stock over the Internet? Did they ever think they'd be able to choose from millions of book and compact disc titles and buy them without ever leaving the comfort of their homes? And (in relation to ColdFusion) how on earth could we make these purchases and be able to track


every movement of the package before it arrives on our front porch? The trick now is to utilize these trends to our advantage. That's what ColdFusion can do for you.

When I was younger – and even now! – my father always said the same thing when I tried to tell him about something great I was doing or about to do: "Don't tell me, show me!"

Well, *CFDJ* plans to one-up that comment. We're not just going to tell you that *CFDJ* is something great. We're not just going to show you, either. We're going to do both. As in our other journals, *Java Developer's Journal* and *PowerBuilder Developer's Journal*, many articles will be followed by the actual codes used by the author. But we're not stopping there. If you check out www.ColdFusionJournal.com, you, as a subscriber to *CFDJ*, can read choice articles and even download the source codes mentioned in the articles. Dad would be proud.

When SYS-CON was founded in 1994, our mission was simple: publish the best technical journals and distribute the most compelling, informative and timely information available anywhere. As we approach our sixth year of publishing quality technical journals, we have the satisfaction of knowing that we're continuing to fulfill this mission.

Today SYS-CON Publications, Inc., is one of America's fastest-growing companies, and we're excited by the opportunities that *ColdFusion Developer's Journal* offers CF and Web developers as a resource that will propel CF development and use to even greater heights.

We want to make this journal one of the most indispensable sources of quality CF information in the world. So here it is. You hold in your hand a collector's item. We also want to hear your questions and comments, so please contact me or any of the editorial staff. It's only through your input that we can create a magazine that's useful, informative and enjoyable. 

About the Author

Chad Sittler is editor in chief of *ColdFusion Developer's Journal*. Chad can be reached at chad@sys-con.com.

chad@sys-con.com

The Object

by Ben Forta

- ✦ There are lots of ColdFusion developers out there – that's good for us.
- ✦ There are also lots of ASP developers out there – that's good for them.
- ✦ ColdFusion developers are empowered with multiple ways to extend the ColdFusion language – that's good for us.
- ✦ ASP developers are not as lucky – that's bad for them.
- ✦ ASP developers rely on COM objects to extend ASP – that's good for us.

That's good for us? Yes, most definitely. ASP has no built-in functionality for many simple things that are built into ColdFusion, so ASP developers use COM objects extensively, even for those little things that we CFers take for granted: generating SMTP e-mail messages, for example. To generate e-mail, ASP developers must install third-party components that provide them with that functionality.

So why is this good news for us? Well, the good news is that there are hundreds of third-party components out there, offering all sorts of functionality. Some are freebies. Others are shareware or commercial software. And ColdFusion developers can take advantage of them all.

This column won't teach you how to write your own COM objects. It will, however, teach you how to use COM objects and how to convert the sample ASP code that accompanies most COM objects into CF code. Note that the examples here use an excellent graphing object, *AspChart*, that allows you to create professional and flexible charts on the fly in several file formats. *AspChart* is created by *ServerObjects Inc.* (www.serverobjects.com). The president of

ServerObjects is Stephen Genusa, one of the leading authorities on ASP development.

Understanding COM Objects

COM, the acronym for Component Object Model, is a software specification designed to allow applications to dynamically link components at runtime. This means that developers can encapsulate functionality into clean, published components and then allow other applications to interact with them. In other words, COM objects are a way to create black-boxed reusable components that other developers can take advantage of.

COM objects are usually DLL or EXE files and can be written in almost any language imaginable, such as C/C++, Visual Basic, Java and Delphi. Once the object is written, it's installed and registered on the computer it will be used on. Registering the COM object publishes it (and its interfaces) to other applications so they can use it.

Almost all COM objects come with installation and registration instructions. Those instructions are the same regardless of the application using the COM object.

Instantiating COM Objects

To use an installed COM object, the first thing you have to do is initialize it for use. Usually this involves instantiating the object.

To initialize COM objects in Cold Fusion, you use the `<CFOBJECT>` tag, which takes the name of the object as an attribute and a name that you use to refer to that object once it's been initialized. The code to initialize *AspChart* looks like this:

```
<CFOBJECT ACTION="CREATE" NAME="Chart"
CLASS="ASPChart.Chart">
```

In this example the `<CFOBJECT>` tag is creating an instance of the chart object. The object's name is "*ASPChart.Chart*" (every object has a unique name) and it is passed in the `CLASS` attribute. `<CFOBJECT>` instantiates the object, which can now be referred to as "*Chart*" as specified in the `NAME` attribute. Just as a point of reference, here's the ASP code that does the same thing:

```
Set
Chart=Server.CreateObject("ASPChart.Chart")
```

As you can see, converting the sample ASP instantiation code to ColdFusion code is pretty simple.

Setting Object Properties

Now that the object is instantiated, we need to pass values to it. COM object values are called *properties*, and every COM object has a different set of properties. Refer to the object documentation for a list of properties and their values.

ct of It All

Two of AspChart's properties are Height and Width. These specify the size (in pixels) of the image to be generated. The ASP code to set these properties is:

```
Chart. Height=300
Chart. Width=500
```

ColdFusion programmers set object properties with the familiar CFSET tag. To set the same two properties with the same two values, your code would look like this:

```
<CFSET Chart. Height=300>
<CFSET Chart. Width=500>
```

As you can see, converting ASP property-setting code to ColdFusion code is also pretty simple.

Setting Object Subproperties

Some properties contain subproperties. For example, AspChart lets you specify the font information to use for the chart title. The ASP code to set the size and font name would be:

```
Chart. ChartTitleFont. Size=20
Chart. ChartTitleFont. Name="Times New Roman"
```

ColdFusion doesn't let you set subproperties directly. Instead, you have to first define a variable that points to the property, then set the subproperties within that new variable. To demonstrate this technique, the following code sets the same two subproperties as the ASP code above:

```
<CFSET ChartFont=Chart. ChartTitleFont>
<CFSET ChartFont. Size=20>
```

```
<CFSET ChartFont. Name="Times New Roman">
```

ChartFont is a temporary variable that points to the Chart.ChartTitleFont property. ChartFont.Size thus points to the Size subproperty within the ChartTitleFont property. Once again, as you can see, with a little effort, converting ASP subproperty-setting code to ColdFusion code is pretty easy to do.

Invoking Object Methods

The last thing you need to know is how to invoke object methods. A method is a function within a COM object. Whereas properties set values within an object, methods actually execute the function using any properties already set. (For this reason, methods are almost always invoked after all the properties have been set.)

Every COM object has at least one method, and some have many more. The code to invoke an object method in ASP looks like this:

```
Chart. SaveChart
```

This invokes the AspChart SaveChart method that actually creates the image file based on all the properties already set. The equivalent ColdFusion code is:

```
<CFSET temp=Chart. SaveChart()>
```


From ColdFusion's perspective, the only difference between setting a property and invoking a method is that methods must have () characters after the method name (methods are essentially functions, and ColdFusion uses () to distinguish functions).

Some methods take one or more parameters. To pass parameters to a method in ColdFusion, just specify them in a comma-delimited list between the "(" and ")" characters. That's all there is to it. Pretty simple indeed.

Where to Go from Here

Now that you know how easily ColdFusion developers can take advantage of COM objects and COM technology, here are some useful sites for you to visit:

- The Microsoft COM Technologies home page is at www.microsoft.com/com. This is a great place to learn more about COM and there are also plenty of links to COM objects and vendors.
- ServerObjects Inc. have created a whole set of COM objects (AspChart is one of many). Their home page is at www.serverobjects.com.
- The CNET ActiveX page at www.activex.com has lists of hundreds of COM objects, with reviews and recommendations as well.

For more information about ColdFusion's COM support, the <CFOBJECT> tag and writing COM objects, see the chapter entitled "Interfacing with COM and DCOM Objects" in my book ColdFusion Web Application Construction Kit (ISBN 07897-14140). 

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 Web Application Construction Kit and the new Advanced ColdFusion 4 Development (both published by Que).

ben@forta.com

The Future (and Past) of E-commerce

Great oaks from little acorns grow



by Jeremy Allaire

Sometimes simple technologies combine in ways that create massive innovation and opportunity. The Web itself is a modern example. Formed as a simple request/response document-delivery mechanism, the core technologies of the Web -- HTML and HTTP -- were central to a broad computing platform shift. It's because these technologies were so simple and accessible that the Web exploded the way it did.

In 1994 we saw that this simple technology could be extended with two other well-known, simple technologies -- scripting and databases. With CFML and a basic database engine, the Web could be used both as a publishing medium and as the foundation for interactive online business. Like the Web itself, technology platforms such as CF have evolved to meet the new demands of online businesses. CF 4.0 represents a mammoth release for Allaire, with major innovations in rapid development, scalable deployment, open integration and total security.

But hidden inside CF 4.0 is another "simplicity revolution." Evolving from the basic need to exchange query data between CF servers, the Web Distributed Data Exchange (WDDX) emerged as a simple technology that solves enormously complex problems.

WDDX is an XML-based technology that enables the exchange of complex data between programming languages, creating "Web syndicate networks". WDDX consists of a language-independent representation of data and a set of modules for a wide variety of languages that use WDDX. (For more technical details or to download the WDDX SDK, check out www.wddx.org.) WDDX and the different WDDX language platform modules are all open-source, freeware technologies and don't require CF.

WDDX improves what you can do with the Web in three major ways: (1) it can be used as the foundation for building Web syndicate networks, where the content and commerce assets of Web sites are exposed and shared to other sites on the network; (2) it increases platform interoperability and legacy migration; and (3) it increases the performance and usability of Web applications.

WDDX can radically change your Web-business model. Until now, most companies and site developers have seen their investments in Web systems and Web sites primarily aimed at end users who access applications through Web browsers. Recently, however, all kinds of companies have begun to realize that their application assets could be leveraged in more substantial ways by exposing them as "services" to other

applications on the Web. Based on Metcalfe's Law, which states that the value of a network increases exponentially with every node added to the network, these companies are building Web syndicate networks where applications serve both end users AND other Web applications.

A good example can be drawn from e-commerce. Any merchant setting up a commerce site will have back-end shipping and fulfillment requirements. This often involves a Web application invoking and tracking a shipment via a provider like FedEx. In the world of Web syndication, FedEx.com is less an end-user site and more an automated portal into the FedEx business computing infrastructure. And the data and services performed with FedEx can be done in an automated fashion using any Web application server or programming language, over HTTP and using WDDX.

In another example, the owners of a Web site offering recipes for Italian food lovers might want to offer additional content and commerce features. Additional content could include popular books on Italian cooking from Amazon.com (including all of the books' details and user reviews) and even a custom user interface for purchasing. All this could be handled under the hood (with the recipe site using ASP, and Amazon.com using CGI and a custom C++ application) with the recipe site providing a custom content and commerce experience that's molded into the user experience.

Web syndicate networks open new business models based on the networked economy. Without a doubt it's the most exciting application of WDDX.

Huge challenges continue to face Web application builders. By their nature, Web applications often involve tying together data from multitudes of disparate systems. They involve integrating with legacy code and applications, even coexisting with desktop fat-client applications based on Visual Basic, MS Access or even Java. WDDX eases this by providing an open, cross-platform and cross-language model for sharing data.

Because WDDX is XML-based, moving complex data between systems can be handled using simple text strings. And since WDDX is available for use with Java and COM, your application data can reach systems built with almost any common desktop programming tool or OS platform. Often referred to as XML Middleware, this use of XML allows you to "tunnel" data between systems, despite differences in implementation and runtime environments. WDDX shields you from having to know anything about XML itself -- all transformations and data exchanges happen under the hood, using things like XML and HTTP as transports for application data.


The platform interoperability advantages are

important to Web developers forced to work in mixed Web application environments. With WDDX you can reuse code and data, preserving your technology investment and supporting richer development and deployment models. This is important to customers who are concerned that an investment in a given platform is closed and can't be used in the future. WDDX provides a conduit to your Web applications that can be reused by other applications.

One limitation of the basic Web architecture is that with an HTML-based user interface you're limited in the richness of interactivity that can be provided to an end user in the browser itself -- HTML is a static user interface. One secondary effect of this is that simple forms of interactivity require additional requests to the server, increasing server load and decreasing the application's performance.

In the past few years the capabilities of the browser increased through the use of JavaScript and DHTML. The 4.0-level browsers have rich programming models that can be used to create flexible and interactive user interfaces. But these have been underleveraged because of browser incompatibilities, and it's been difficult -- if not impossible -- to move application data between Web browsers and application servers. So despite an increase in the capabilities of Web clients, even simple data binding has been impossible.

WDDX provides a simple, lightweight cross-browser solution to increasing usability and performance in your Web applications. Because JavaScript supports WDDX, you can easily move data between any server-based applications and browser-based applications using JavaScript. In this model you could have a database query on the server passed to a JavaScript browser where it works with the data locally, rendering it using the Document Object Model and DHTML, then passing any changed data back to the server when it's completed. The WDDX SDK includes extensive examples for using WDDX to enhance DHTML user interfaces and to create data-binding and offline browser applications.

I strongly encourage you to check out WDDX via the project homepage (www.wddx.org), and let us know what you think. 

About the Author

Jeremy Allaire is a cofounder and vice president of technology strategy at Allaire. He helps determine the company's future product direction and is responsible for establishing key strategic partnerships within the Internet industry. Jeremy has been a regular author and analyst on Internet technologies for the past seven years, and he holds degrees in both political science and philosophy from Macalester College.

jeremy@allaire.com

Eprise

www.eprise.com

LEVERAGING COLDFUSION: Putting Client/Server on the Web PART 1

Replicating the User Interface and Functionality

by Jerry Bradenbaugh

This article is the first in a series of three focused on using ColdFusion to replicate client/server application functionality across an intranet, extranet or the Web. It discusses replicating the look and "feel" of a client/server application from a user interface perspective.

One of our clients came to us with a problem. The client had a frumpy but effective fulfillment request application written in MSAccess. They wanted to convert it to an intranet application to (1) increase user access and scalability, (2) improve performance time and (3) maintain the same look on the intranet as it had as a client/server application. Enter ColdFusion.

In the next few pages we'll take a closer look at how you can use ColdFusion to meet your client's business requirements across an intranet, extranet or the Web by converting a client/server application to a Web-enabled application. I'll show you how the Red Oak Technologies development team

did so by examining a portion of the application Red Oak developed called the Fulfillment Request System. You'll see screen shots of the interface and much of the code behind it.

As you read on, remember that ColdFusion is designed to easily accommodate other technologies, so we'll be covering HTML, JavaScript, ODBC and SQL statements. If you're not up to par on these technologies, keep an online reference close by.

The links in Table 1 should help:

Technology	Site Name	URL
HTML	World Wide Web Consortium	http://www.w3.org/
	ZDNet: HTML User Garage	http://www.zdnet.com/products/htmluser/garage.html
JavaScript	Netscape's DevEdge	http://developer.netscape.com
	HotSyte: The JavaScript Resource	http://www.serve.com/hotsyte/
SQL	Introduction to Structured Query Language	http://w3.one.net/~jhoffman/sqltut.htm
ODBC	Microsoft Data Access Components	http://www.microsoft.com/data/reference/odbc2.htm

Table 1

The FRS Overview **Business Challenge**

The Web-enabled application must allow users to process customer orders (taken by phone or e-mail) via their intranet. Since the representatives take the broker orders in a fast-paced environment, the application must have a clean (graphic-free), fast interface. The application must also look similar to (if not better than) the client/server version that representatives use to easily modify or add customer information and send orders to the shipping department.



The left column of the screen displays the products available in the Materials Available SELECT list and identifies the product source in the Group SELECT list. The middle column allows the user to add and remove products from the request, choose product quantity and save the order request. The right column displays an itemized list of the current products to order. Figure 3 shows selected product information in the right column.

When the user is satisfied with the selected products, he or she chooses the SAVE button. This opens the Shipping Entry screen, where the user can similarly add or edit shipping information and save the information for the shipping department. Figure 4 shows the Shipping Entry screen with a populated shipping form, ready to go.

This application employs the following techniques:

- Generating HTML on the fly
- Processing user events/generating dynamic JavaScript
- Using ColdFusion template files
- Database access

This should be fun. Let's get started.

Generating HTML on the Fly

A key feature of almost any server-side scripting is generating custom Web pages according to any number of variables: time of day, month, the visitor's referring IP address, the browser or your mom's favorite yogurt flavor. Okay, I'm reaching there, but you can make small or enormous adjustments in content depending on a multitude of things. ColdFusion is no different. Consider the HTML form shown in Figure 1.

It looks pretty straightforward; just a big form with a handful of fields, right? Well, yes and no. *Yes* to the words "big form with a handful of fields"; *no* to the word "just." It's more than that. The same page is called when the user has opened the application for the first time or when he or she has selected a broker from a search results list (see the right column of Figure 1).

Opening the application for the first time is easy. Just pass in the static HTML form setting all form element values to empty strings or the default OPTION of the SELECT lists and be done with it.

However, things change if the user has already performed a search and selected the broker he or she wants from the results list in the right frame results list. We'll still print the same form fields to the screen, but the values of the form fields will be populated with information from the broker records in the database instead of from empty strings.

Listing 1 displays the code that sets

things up before we generate any HTML. It comes from file CustInputFrame.cfm.

In the second block of code the local variables start with the same value: an empty string. If the user calls the page for the first time, CF variable #custno# retains its value of 0. Here it is again:

```
<CFPARAM NAME="custno" DEFAULT=0>
```

However, if the user calls this page by clicking on the link of a client record, #custno# will be set to something other than 0. That kicks off a database query (covered in more detail later).

```
<CFIF #custno# IS NOT 0>
<!-- Read Customer File
Blank if new entry
Found - (calling from 'EDIT CUSTOMER INFO'
button of Shipping Entry)
-->
<CFQUERY NAME="custlist"
DATASOURCE="#CLI.ENT.datasource#">
select * from vwcustomer
where personID = #custno#
</CFQUERY>
```

It then replaces the empty string value of the local variable with data recovered from the query. An example of this can be found in Listing 2.

These local variables, whether they contain empty strings or otherwise, are then used to populate the form in the Customer Entry/Update Screen. The rest of CustInputFrame.cfm is basically an HTML form. Here is a portion to get an idea. The bold code in Listing 3 indicates a CF variable in use.

Processing User Events/Generating Dynamic JavaScript

Okay. Cool. The HTML form is in the user's face. In designing the application, we tried to maximize user interactivity by processing as many user events as possible. Almost all user events in the application initiate some kind of page or database update. FRS is a multiframe application, which gives the application more "seamless" screen updates. That is, instead of printing the output from a .cfm file directly to a frame the user sees, the content is first targeted to a hidden frame. When the content is finished, the output is passed from the hidden frame to the "user" frame. Four user events initiate this.

- Clicking form buttons (BUTTON, SUBMIT and RESET)
- Clicking links
- Form field text changes
- Double-clicking

There are no real surprises with the first two, but let's go over them anyway.

Business Solution

FRS is a customer information-capturing and product-ordering and -shipping application for financial clients called *brokers*. Client representatives use this application to easily capture broker information and process broker requests for financial product brochures. Brokers can request information packages and brochures about CDs, IRAs, bonds, mutual funds and the like. Figure 1 shows an opening look after the login.

When brokers contact client representatives by phone to make a request, the representatives can easily search, access and modify broker information. Broker profiles can be searched by almost any single piece of data – first name, last name, city, phone number, etc.

Once the representative accesses and selects the correct broker profile (and updates information, if necessary), he or she can handle the broker's request for financial products. This happens in the Material Entry screen (see Figure 2).

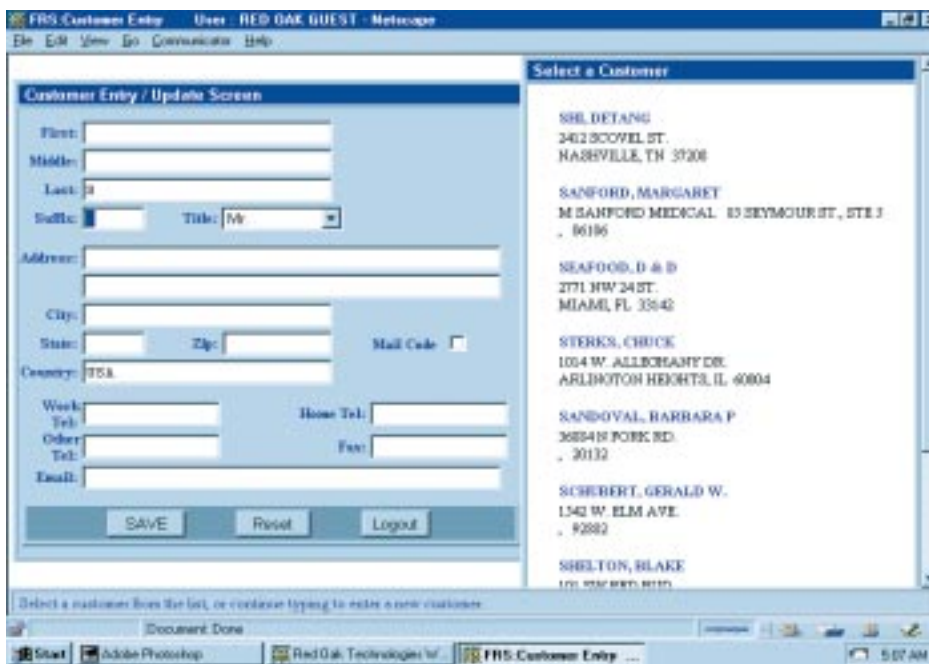


Figure 1

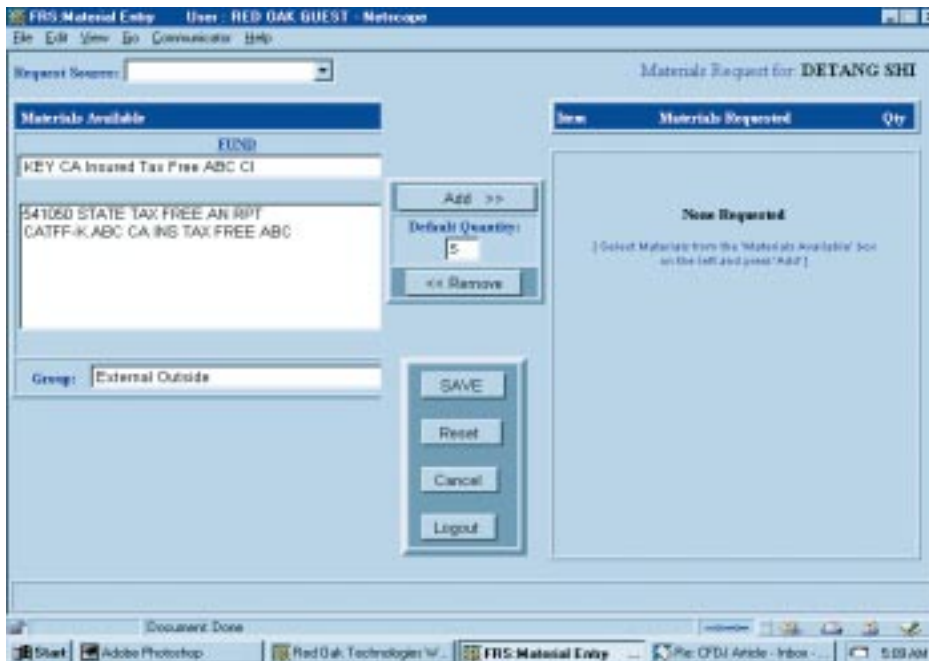


Figure 2

Clicking Form Buttons

On the Customer Entry/Update Screen in Figure 1, users can create, modify or reset customer information and log out from this screen. Each button makes a call to a corresponding ColdFusion file that changes client information or ends the user session.

Clicking Links

Users can generate client search results in the right column of the Customer Entry/Update Screen. Each result contains a .cfm link that populates the form on the left side of the page.

Form Field Text Changes

Did you notice that there is no button

or graphic to submit a client search? FRS searches the client database by last name. When a user enters text in the Last: field and then places focus elsewhere on the screen (using the TAB key to advance to another form field, for example), the JavaScript onChange event handler evaluates the entered text to see if it has changed and, if so, calls file LookUpName.cfm to search the database for matches. Here is the code for the Last: field:

```
<input type="TEXT" name="LastName" value=""
size="25" onchange="checkname()"
onFocus="selectContents(this)">
```

JavaScript function checkname() calls LookUpName.cfm to look for matches, like so:

```
function checkname() {
top.sub.location="LookUpName.cfm?lastName="
+ document.CHKNAME.LastName.value;
return true;
}
```

The value of field LastName is concatenated to the query string of LookUpName.cfm. We'll get into the database goodies a bit later. For now, just remember how the search operation is called.

This is great for existing clients, but what about creating new clients? The user can't search for clients that aren't there; he or she must first create the record by entering information field by field until it's finished (i.e., he or she chooses the SAVE button). To reduce errors, FRS performs a similar validation on the text entered in the Zip: field. If the zip code is associated with multiple cities, the user is prompted to choose from those cities. Using the onChange event handler again, JavaScript function validatezip() gets the call. Listing 4 contains the corresponding code.

The important thing to take away from this is that if the user properly entered a valid zip code, function lookupzip() is called to format the zip into the query string for another database search. Otherwise, StatusBar.cfm is called to alert the user that he or she entered an invalid zip code format (that is, the user entered something that couldn't possibly be a zip code).

This status bar is not the browser status bar, but one we've created using a table with a grayish (all right, silver) background. This table is "rewritten" many times throughout the user experience to alert the user of an error or instruct him or her about the next step). StatusBar.cfm is in Listing 5.

It seems as if there's not much here – just an HTML table with a single row and data cell. That's true, but this file is called whenever the Customer Entry / Update Screen is called and after every search. Therefore, we use the CFPARAM tag to assign default values to variables lerror (the error message text) and color (the text color of the error message text). The default values are an empty string for lerror (""), and the string Navy for color. That way nothing is printed to the screen on the opening call except an empty table with a silver background.

If the zip code is formatted correctly, function lookupzip() gets the call. As you can see from the code in Listing 6, ColdFusion file LookUpZip.cfm will initiate the database search for matching zip codes.

Schlumberger

www.cyberflex.slb.com

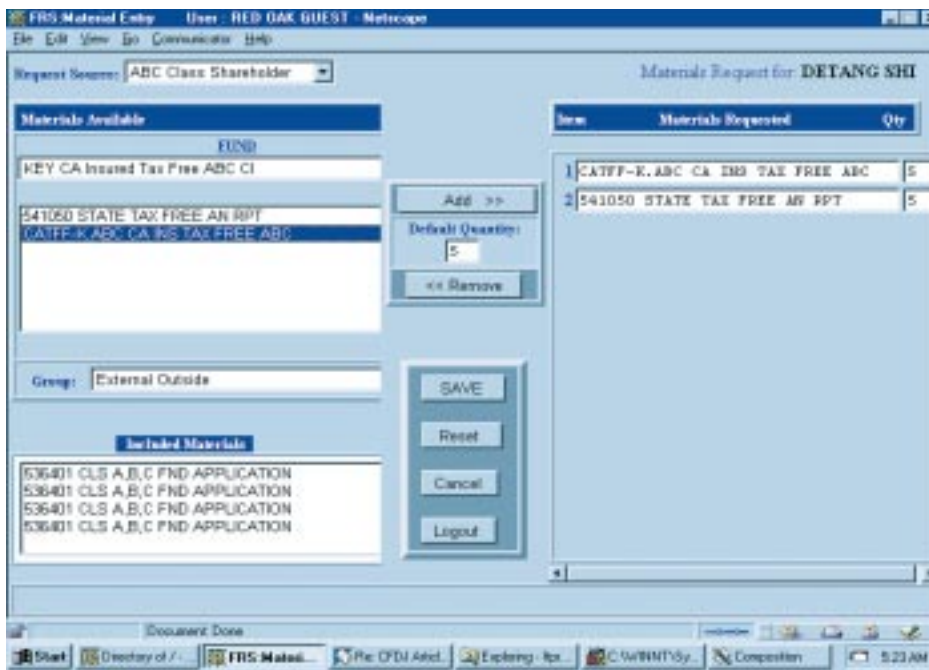


Figure 3

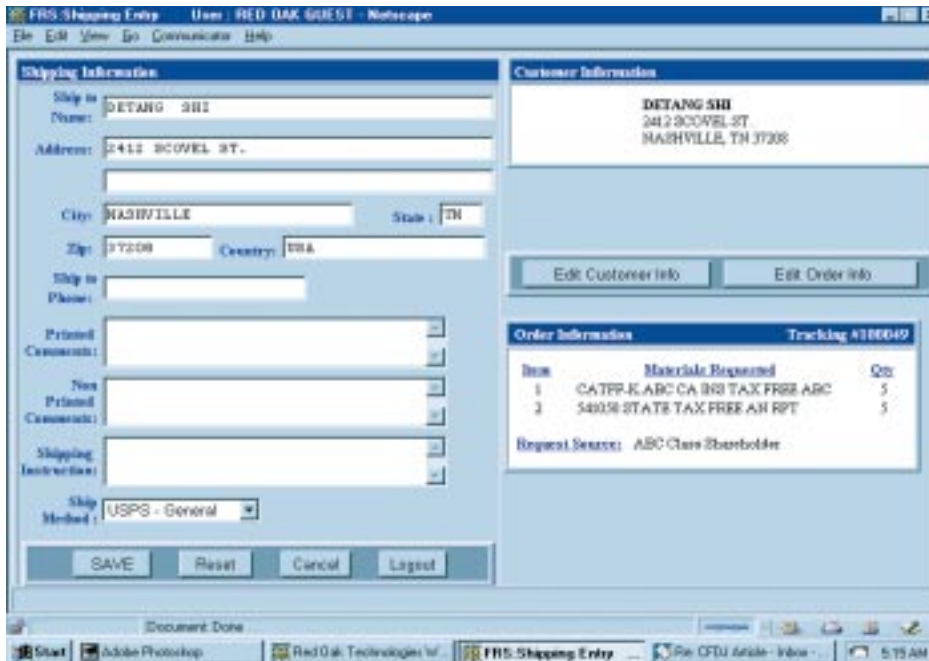


Figure 4

The query string developed here is somewhat more complicated than it is for the last name search. Not much has really changed, though. LookUpZip.cfm still initiates a database search based entirely on the text entered in the Zip: field. The other three parameters represent the frame and form names and the name of the frame containing the "status bar" display discussed earlier.

These three parameters identify the form in the browser object hierarchy. This directs the output of LookUpZip.cfm to the correct form to populate. In other words, JavaScript is going to populate a form with any results the search found. These variables identify in what frame the form is located and the name of the form. The code

for LookUpZip.cfm is rather long, so I'll just include some of the more interesting parts.

```
<SCRIPT LANGUAGE="JavaScript 1.1">
<!--
/* Assign URL variables to JS variables
*/
<CFOUTPUT>
var framename = #framename#
var formname = #formname#
var formtype = "#formname#"
var statusname = #statusname#
</CFOUTPUT>
...
...
...
```

</SCRIPT>

This is where dynamic JavaScript comes in. Until now, the user has been getting HTML generated on the fly. Not bad, indeed, but the code above shows JavaScript variables being set to the value of ColdFusion variables. This is dynamic JavaScript. These ColdFusion variables happen to be the same ones with values in the query string created by function lookupzip().

Double-Clicking

This feature doesn't reveal any secrets about using ColdFusion, but rather about the way that the .cfm file is called. This isn't a required feature, but it adds a nice touch. Figure 2 displays the available financial products. Users can add products to the client list in two ways. Single-clicking on a product from the left-hand SELECT list, then choosing the ADD >> button, OR simply double-clicking any item in the SELECT list. Again, this doesn't add any miraculous CF logic, but look at the way it's called. Listing 7 shows the opening SELECT tag.

Once the user clicks, he or she has a specified number of milliseconds (in this case, 100) to click again. If that happens, function displayinfo() is called to update the list of requested materials on the right-hand side.

Using ColdFusion Templates

This section is considerably smaller than the others, but no less important. Many of the JavaScript functions of FRS are used in more than one place. For that very reason, several .cfm files have the following code:

```
<!-- External JS program for all functions -->
<cfinclude template="js_Customer.cfm">
```

File js_Customer.cfm is a file of static JavaScript functions. ColdFusion templates are a great way to economize on frequently used code. It's also great for code maintenance. Suppose you had those JavaScript functions hard-coded in five different pages. If you want to make even one change, you have to make at least five changes.

Some argue that the code could be dumped into a JavaScript source file, like this:

```
<SCRIPT LANGUAGE="JavaScript 1.1"
SRC="js_Customer.js"></SCRIPT>
```

True, but this has two disadvantages. First, since the code is parsed in the browser, the client is making the source file request. That's an extra HTTP request you

Live Software

www.cfanywhere.com

could have saved by using a CF template on the server. Second, if you worried about backward compatibility, most browsers with support for JavaScript1.0 don't support JavaScript source files (though some MSIE3.x versions do). With CF that's not an issue; it's all on the server side.

Database Access

Ah, yes...the database. We find ColdFusion makes it particularly easy to query existing databases via ODBC. FRS uses an MS SQL server database. There are three database calls within Figures 1 and 2. Whether INSERTing or SELECTing, all three are essentially the same, so we really need to look at only one. Figure 1 displays the output of file CustInputFrame.cfm, the one that populates the Customer Entry/Update Screen with client information after the user performs a search.

```
<CFQUERY NAME="custlist"
DATASOURCE="#client.datasource#">
    select * from vwcustomer where personID =
#custno#
</CFQUERY>
```

The <CFQUERY> tag assigns custlist as the name of the query and the datasource to client.datasource. Variable client.datasource is a client-state parameter. Client-state management is the ability to track users as they move through an application. You can enable client-state management in your application, as does FRS, by including the <CFAPPLICATION> tag in your .cfm file.

```
<CFAPPLICATION NAME="FULFILLMENT" CLIENTMAN-
```

```
AGEMENT="YES">
```

This tag makes the .cfm file in which it is contained eligible for client-state management. If you add this to your Application.cfm file, all .cfm files of the application can use client-state management (Part 2 of the series will cover client-state management in more detail). Our Application.cfm file includes the following:

```
<CFSET #client.datasource#"DEMO_FRS">
```

This associates the client variable to the database named DEMO_FRS, the datasource name associated with the SQL database in the ODBC administrator. With the query named (custlist) and the database identified (DEMO_FRS), all that remains is a SQL statement dictating what to return from where.

```
select * from vwcustomer where personID =
#custno#
```


Since we want all the client information to populate the HTML form, we use the syntax select * . The records are in table vwcustomer. Each client has a unique customer number (#custno#) that's stored in column personID. Remember that the value assigned to #custno# was passed in via the query string when the user clicked on the search result link.

What happens to the data recovered from the database query? Each of the local variables originally set to an empty string in CustInputFrame.cfm is set to the value of the data returned from the query, as seen in Listing 8.

As you saw previously, the value of the HTML form elements then assumes the value of all the local variables. By the way, variable #t_mailcode# isn't used until the shipping portion of the application, so its value is set to an empty string.

Putting It All Together

As it turns out, our client was pleased with the application. Putting the application on the intranet increased user access, improved response time and still acted like the original client/server application. ColdFusion's role as an application server enabled a straightforward deployment and easy application management while remaining transparent to the user. Consider how your organization can use CF to do the same. Combining JavaScript techniques on the client side with ColdFusion makes a unique duo that can take your client/server app to the Web.

Parts 2 and 3 of this series will demonstrate how you can administer security and timed events of your applications, and how you can use CF function for validation and more back-end processing, respectively. 

About the Author

Jerry Bradenbaugh works as a senior Web developer for Red Oak Technologies, Inc. (www.redoaktech.com). He focuses on robust Web-enabled systems using JavaScript, ColdFusion, Java and database technologies. He is also the Webmaster of HotSyte - The JavaScript Resource (www.serve.com/hotsyte) and is currently writing a JavaScript book for O'Reilly & Associates, Inc.

 jerryb@redoaktech.com 

Listing 1.

```
<CFAPPLICATION NAME="FULFILLMENT" CLIENTMANAGEMENT="YES">
```

```
<!-- Declare URL Variables -->
<CFPARAM NAME="orderid"      DEFAULT=0>
<CFPARAM NAME="custno"       DEFAULT=0>
<CFPARAM NAME="source"       DEFAULT="">
<CFPARAM NAME="group"        DEFAULT=0>

<!-- Declare local variables and initialize -->
<CFSET #t_firstname# = "">
<CFSET #t_middlename# = "">
<CFSET #t_lastname# = "">
<CFSET #t_prefix# = "">
<CFSET #t_suffix# = "">
<CFSET #t_address1# = "">
<CFSET #t_address2# = "">
<CFSET #t_city# = "">
<CFSET #t_state# = "">
<CFSET #t_zip# = "">
<CFSET #t_country# = "USA">
<CFSET #t_mailcode# = "">
<CFSET #t_wphone# = "">
<CFSET #t_hphone# = "">
<CFSET #t_ophone# = "">
<CFSET #t_fax# = "">
<CFSET #t_email# = "">
```

```
<CFIF #custno# IS NOT 0>
<!-- Read Customer File. Blank if new entry -->
<CFQUERY NAME="custlist"
DATASOURCE="#CLIENT.datasource#">
```

```
select * from vwcustomer
where personID = #custno#
</CFQUERY>

<!-- Set local variables to record found -->
<CFOUTPUT query="custlist">
<CFSET #t_firstname# = #firstname#>
<CFSET #t_middlename# = #middlename#>
<CFSET #t_lastname# = #lastname#>
<CFSET #t_prefix# = #prefix#>
<CFSET #t_suffix# = #suffix#>
<CFSET #t_address1# = #mailtoaddress1#>
<CFSET #t_address2# = #mailtoaddress2#>
<CFSET #t_city# = #mailtocity#>
<CFSET #t_state# = #mailtostate#>
<CFSET #t_zip# = #mailtopostalcode#>
<CFSET #t_country# = #mailtocountry#>
<CFSET #t_mailcode# = "">
<CFSET #t_wphone# = #phonework#>
<CFSET #t_hphone# = #phonehome#>
<CFSET #t_ophone# = #phoneother#>
<CFSET #t_fax# = #fax#>
<CFSET #t_email# = #email#>
</CFOUTPUT>
</CFIF>
```

CODE LISTING

The complete code listing for this article can be located at www.ColdFusionDevelopersJournal.com

Allaire 2

www.allaire.com

What's Different About ColdFusion's Version

4.0

Some of the enhancements that make the upgrade worthwhile for developers

by Richard Schulze

Recently Allaire released its newest version of ColdFusion, their powerful database integration tool. Part of the 4.0 release includes an addition to the Allaire lineup – the Enterprise version. This release hit the public later than planned, but was worth the wait. As I've been involved in this release since its alpha builds, I can attest to the fact that the development team took their time and listened to their control group of testers.

This article will give an overview of some of the improvements that 4.0 gives the developer without getting bogged down in technical jargon. As lead developer at Motorola for an application that's being used nationwide – soon to be worldwide – I'm interested in things that make my code cleaner, faster and more powerful. The following text assumes you need the same for your applications, and those are the enhancements that will be discussed.

Before diving into these enhancements, I need to point out the major difference between the Professional and Enterprise versions. While both can be considered safe for mission-critical systems, the Enterprise version takes it to the extreme. Allaire struck an apparent agreement with a company called Bright Tiger and their product, Cluster Cats. Cluster Cats provides the user with the ability to load-balance between multiple servers, provide server fail-over support and monitor various services on the NT platform. With an upgrade to Bright Tiger's full product, you'd also get content replication. Even without the upgrade, with a little imagination you could provide your own content replication through the use of batch files and the AT command set and NT's scheduler.

Don't assume that the Enterprise version is the only product in Allaire's lineup that can be considered mission critical. With the enhancements that were made to the product, I have had great reliability with the Professional version on Microsoft's IIS 4.0 platform. The use of Enterprise may eventually be forced upon me as the application I'm working on moves to a worldwide audience, as traffic is increased and as load balancing is required. A few of the enhance-

ments that benefit both speed and reliability are obvious when looking at the ColdFusion Administrator for the first time.

The Administrator seems much more complex than the previous version...and it is. This is not because someone forgot about the end user, it's simply a statement that this product now packs all of the security, enhancements and flexibility required of today's database-centric applications. One of the first things you'll notice is the option for unresponsive requests. If you have set a time-out value for requests and those requests start timing out, the service is stopped and restarted. This singular feature is a powerful enhancement in itself. As a previous owner of a small ISP, I often had to restart services because things were locked up. This one feature, along with IIS 4's ability for process isolation, saved an awful lot of headaches and phone calls.

The features continue with a new one called the *trusted cache*. On first look, it impressed me as a feature that would never be used. On second look it was perfect for my needs. What a trusted cache does is to "trust" you, the administrator, that all the files in the template cache are the right ones and that no inspections for updates have to be made. Less work is faster! As I do all of my development on a test system and then upload the final code to our main server, I realized that this feature seemed perfect for an intranet that you have direct control over. This would cause a nightmare for an ISP, requiring the service to be stopped and restarted. The same workaround for the Access ODBC problem of using the NT scheduler to stop and restart the service (`\cfusion\bin\cycle.bat`) might ease this issue with ISPs.

There are other enhancements on what can be done with queries and such that enhance speed, but they'll be discussed with tag enhancements. Without getting bogged down in the aforementioned technical jargon, let me emphasize that Allaire has done a lot of work in the background to make their product line faster and more stable. They now boast "Enhanced Multithreaded Service" that scales well across multiple processors and "Improved Thread Pooling," which would probably be better discussed in a white paper than here. The bottom line is, it's faster and more reliable.

In fact, there are so many new features in this release that it will be difficult, if not impossible to cover them all. The best place to start a discussion on a new product would be with installation concerns.



Being an MCSE/MCT, I lean pretty heavily toward Microsoft products, so I'll discuss two issues regarding the installation of ColdFusion on the IIS 4.0 platform.

Often, depending on your upgrade path of ColdFusion and an Internet information server, you'll have multiple, redundant application mappings set, which has been known to cause problems. But the condition is easily remedied. The first step would be to open the Microsoft Management Console (MMC) (see Figure 1).

Right-click on the server name and go to Properties (see Figure 2). Click on Edit with the WWW Service highlighted (default). Click on the Home Directory tab and then the Configuration button. This will bring up the Application Configuration interface. You should have only one mapping for .cfm files and one for .dbm files. In Figure 3 you'll notice that there are two of each; this is typical. Simply highlight one and click on Remove. Repeat for the other duplicate entry. It would be nice to do a multi-select, but it just won't work that way. Once the duplicates have been removed, double-click on each of the entries and ensure that the script engine Checkbox has been selected. Normally it is, but it's been known to cause a problem when it isn't.

The second installation issue is that of Crystal Reports. ColdFusion has direct support of Crystal Reports, and it's still in widespread use. When Crystal Reports 6 or 7 are installed onto the server, its default installation directory is \Seagate Crystal Reports. If the installation/upgrade is allowed to finish in this configuration, you'll quickly find out that Crystal Reports won't work. The problem is not with Crystal Reports or ColdFusion, but with IIS. Following the installation you'll see an entry in the Application Configuration (Figure 3) dialog for .rpt files. Double-clicking on it will bring up the Add/Edit Application Extension Mapping interface. If you then click OK, you'll see that IIS 4 doesn't understand the two spaces in the folder name and it will give you an error.

Correcting the misunderstood directory can be done in two ways. The first is to manually remove the spaces from the folder name with Windows Explorer and then to use REGEDIT (not REGEDT32) and do a search for "Seagate"; then change all of the entries in your registry to reflect the new folder name. Please keep in mind that making alterations to the systems registry is potentially devastating to a computer system. The second and preferred method is to simply uninstall Crystal Reports and reinstall into a directory without the double spaces. The service pack for IIS doesn't fix this problem...sorry.

The real joy of this upgrade is for the developer. While it's true that all of the enhancements already discussed have been geared toward the ColdFusion environment, there are plenty still left for the developer. The first enhancement I noticed right off was the debugging. It seems as if it's become more accurate. You'll notice that the error messages that usually directed your attention to the general area of the problem now seem to be more accurate. Better debugging means faster development.

Though this upgrade added many new tags and functions, a good portion of it didn't, enhancing current ones instead. Some of the changes were minor; others were more dramatic. The CFABORT tag is one of the simplest examples of this. Often my programming effort is geared toward data validation, making sure that what is being drawn out and going into the database is what is allowed and correct. This often requires complex server-side validation routines. If these routines fail, a message must be displayed to the user. The CFABORT tag now has an optional attribute, showerror. This option allows the programmer to put the error right into the tag instead of calling out a separate template, or to manually add the HTML code prior to the CFABORT tag. For example:

```
<CFIF Password NEQ PasswordConfirmation>
<CFABORT SHOWERROR="You have not entered the same password in both
fields">
</CFIF>
```

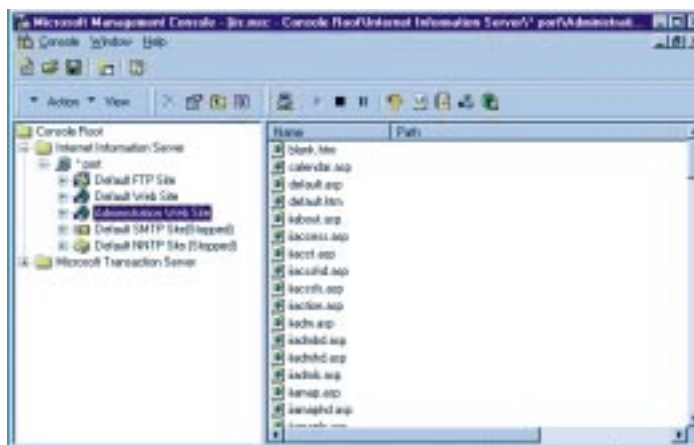


Figure 1

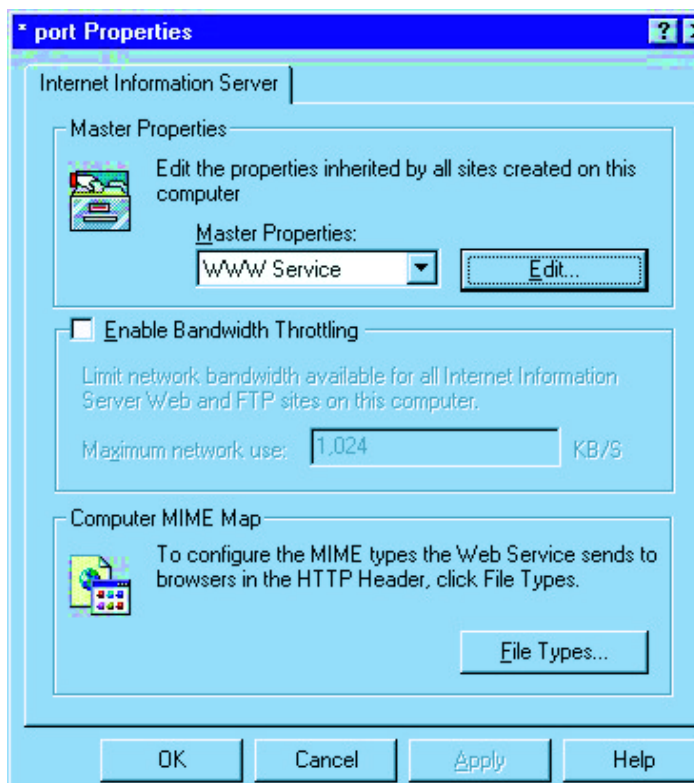


Figure 2

This script would check the values of "Password" and "PasswordConfirmation"; failing that validation would result in the message pre-defined in the SHOWERROR attribute being displayed in the error page.

Error Occurred While Processing Request

Error Diagnostic Information
You have not entered the same password in both fields

In some cases you may want to dress up your error messages; I have adopted the above method as my standard. It is effective, and cleans up my code quite a bit.

The next tag enhancement that has taken a load off the server has to do with the CFQUERY tag. This tag has undergone a major transformation in my estimation. The attributes available to this tag are numerous:

BLOCKFACTOR, CACHEDAFTER, CACHEDWITHIN, DATASOURCE, DBNAME, DBSERVER, DBTYPE, DEBUG, MAXROWS, NAME, PASSWORD, PROVIDER, PROVIDERDSN, TIMEOUT, USERNAME

Several of these options have always been around and I'm not going to explain every one. BLOCKFACTOR, CACHEDAFTER, DBSERVER, DBTYPE, PROVIDER and PROVIDERDSN all offer flexibility and power this tag never had before. The one option I have found very useful is CACHEDWITHIN.

Before I discuss what this new option offers, perhaps a little background is in order. Often in working with databases you'll find numerous "lookup" tables. A lookup table is one that associates a number with a larger group of information or perhaps its text counterpart. For example, a state lookup table would associate an int (number) with the standard abbreviation and full text name of the state. Everywhere else that state information is stored, the number goes in those tables, not the text.

Why would you do this? Simple. Searching on numbers is faster than searching by text. If your application provides the user with the ability to search through thousands of records using the state as possible search criteria, those searches will be quicker if you search by numbers. This is a simple but effective speed enhancement.

When I initially look at a database structure, one of the first things I do is to see where lookup tables are needed and how I can combine them. The application I'm working on now had no less than a dozen lookup tables that I was able to combine into a "common lookup" table. I then placed all this information into server variables with the following code:

```
<CFIF IsDefined("Server.GetCommonData") IS "No">
<CFQUERY DataSource="MyDS" Name="Server.GetCommonData">
  SELECT * FROM tblCommonLookup ORDER BY LookupType
</CFQUERY>
</CFIF>
```

Throughout the remainder of the application I simply used the information that's stored in memory (server variables). I effectively eliminated thousands of queries per day with this one bit of code in my login action page. What does this have to do with CACHEDWITHIN? Not a lot...and everything.

Not all lookup information explained above will fit into a set model. In certain parts of an application, queries that never change are used. At these times I use this new option – when a query is used time and time again without the data ever changing. When using CACHEDWITHIN, you'd normally use CREATETIMESPAN to set how long the query is to be cached. For example:

```
<CFQUERY datasource="MyDS" name="CacheThis"
Cachedwithin="#CREATETIMESPAN("0", "1", "0", "0")#">
  SELECT ....
</cfquery>
```

The above code would cache the query for a period of one hour from the time the original query had taken place. You'll see this in your debugging code. You need to be careful with this option because if you're planning on adding new information to the affected table(s), it won't be reflected until the cached query times out and the information is drawn from the database again and not from memory.

Another major change is that of security. Security has been a major focus point of the entire 4.0 release. A major help to ISPs is the ability to control the CFFILE tag. Actually, CFFILE just follows the security guidelines you set up as an administrator. Even without setting up any security, CFFILE is not able to upload files or make any other file manipulations anywhere outside your Web server's directory structure. In version 3.x you could upload anywhere. I used to show this to students when I taught ColdFusion for Allaire just to demonstrate the potential hazards it posed. All of that has changed now.

During install you'll have the option of selecting the advanced

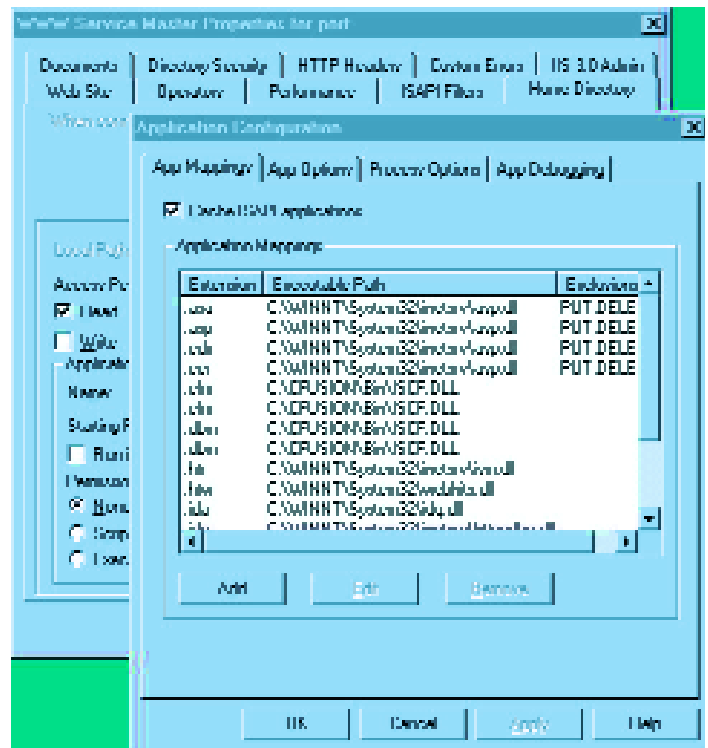


Figure 3

security options; if you do, you'll be required to install Microsoft Active Directory Services Interface (ADSI) onto your server. To discuss the specifics on how to set up various security schemes would be an article in itself. If you're running an ISP or even an intranet with various applications, it would be worth your while to investigate this and the Studio integration with these various security measures.

Although I don't have the space to discuss every enhancement I've learned to use to empower myself as a programmer, this article would be incomplete without mentioning ColdFusion Studio. Studio 4.0 also received a major rework. Some of the new features that come to mind are the ability to assign Hot-Keys to code snippets, better integration into the server and security features, improved project management, faster page loading, much faster find and find/replace features and a host of others. I have been absolutely thrilled with Studio since its release. Although its query builder is a fantastic tool that handles a good portion of my SQL effort, it could use a bit of a boost. I have used Crystal Reports 6's visual builder to write better code for some very complicated queries. If you're using HomeSite instead of Studio, you're doing yourself an injustice.

The most profound part of this Studio release is the debugger. In Studio you can now set breakpoints in your code, check variables and values, look at the stack and more. Studio has now separated itself not only from other HTML editors, but from the basic definition of an HTML editor. It is now a programming tool for applications, not just pretty Web pages. rschulze@finalbitsolutions.com

About the Author

Richard Schulze is the owner and operator of Final Bit Solutions, which specializes in Network/Internet consulting and training. His experience ranges from C and Visual Basic programming to formal network training and experience leading to an MCSE/MCT. He is an instructor at the University of Southern California and head programmer at Motorola for a major intranet application utilizing ColdFusion. You can reach Richard at rschulze@finalbitsolutions.com.

Allaire 3

www.allaire.com



The Data Path: From Database to Web Page and Back Again

Using SQL, HTML and JavaScript to retrieve data

by Andy Birchall

ColdFusion is basically a language that acts as an intermediary between databases and the exciting World Wide Web. As such, it has the marvelous ability to retrieve data from databases (or “datasources,” as they are defined) and place it on the page in whatever form the developer wishes, whether just a list of clients, a current menu for a restaurant or major applications such as a hotel booking system.

To help it accomplish this task, CF works with two other major elements; occasionally, to add finishing touches, a third is employed. These other agents are of course SQL, or structured query language, and HTML, or Hypertext Markup Language (to demystify a couple of the many acronyms present in the world of CF). The finishing touches are often applied with a touch of JavaScript.

This article aims to have a look at some of the basics involved in retrieving and handling the data, hopefully pointing out some of the more common pitfalls that are often encountered in the rocky path to writing that application with a deadline rapidly approaching.

The various steps in data manipulation will be looked into, beginning with the use of SQL to request the data in the first place, some CF hints and tips, followed by a small dose of JavaScript.

How Do I Request the Data I Require?

Your data is, of course, safely locked away in nicely organized tables linked by appropriate key fields, and it has those handy indexes set up to keep everything speeding along. (Well, that IS the theory anyway.)

CF asks for the data by using the <CFQUERY> tag. This nicely encloses a piece of SQL code that talks to the database on the server (the databases are all registered as “datasources” on the CF Administrator page), and the database responds by sending back the information requested.

Many different database applications are used commercially – dBase, Access, FoxPro and SQL Server, for example. Now SQL was originally intended to talk to databases and ask for information from them. There is a standard core that all the database applications use. Each one has its own peculiarities, however, and the SQL you use will in some minor ways be specific to the database you’re storing the data in. SQL Server, for example, has a wonderful CONVERT function that lets you switch between data types and is especially useful for pre-formatting dates in the table. But try this function with a FoxPro database and you’re out of luck.

The core elements of SQL are reasonably easy, however, and were originally written to appear as if a “spoken” request was being made. Let’s have a look at a couple of examples here:

- a. Can I have all the ClientNames in the Clients table, please?

```
SELECT ClientNames
FROM tblClients
```

- b. Can I now order them alphabetically, in ascending order?

```
SELECT ClientNames
FROM tblClients
ORDER BY tblClients ASC
```

- c. Now I want just those clients who have surnames beginning with “C”, but I also need their mailing address, telephone number and e-mail address. It would be nice to know which company they are connected with and what their name and telephone number are as well.

```
SELECT tblClients.ClientNames,
tblClients.address, tblClients.tel,
tblClients.email, tblCompanies.address,
tblCompanies.tel
FROM tblClients INNER JOIN tblCompanies ON
tblCompanies.CompanyID = tblClients.CompanyID
WHERE tblClients.ClientNames LIKE 'C%'
ORDER BY tblClients.ClientNames ASC
```

A small note of explanation on the LIKE ‘C%’ bit: the LIKE operator is used for matching strings. Now the % used here is a wild card, but again, different database applications use different characters for the wildcard. Access uses ‘*’, SQL Server uses %, so make sure you use the right one for your database or you might not see the data you expect!

Example c. also introduced the idea of grabbing data from two tables (and can be extended to a much more complicated table structure). Again, other syntax can be used. When the tables are joined by one field having the same value in both tables (the INNER JOIN above), you can also just put this in as an extra WHERE condition so the query would now say:

```
SELECT tblClients.ClientNames,
tblClients.address, tblClients.tel,
tblClients.email, tblCompanies.address,
tblCompanies.tel
FROM tblClients, tblCompanies
WHERE tblClients.ClientNames LIKE 'C%' AND
tblClients.CompanyID = tblCompanies.CompanyID
ORDER BY tblClients.ClientNames ASC
```

Now, let’s think of retrieving all clients who have logged into our system over the past month. Here we need to use some means of defining what the period of the last month is. This is one of the means by which CF can come into play, as we can utilize CF functions within the <CFQUERY> SQL block.

```
SELECT tblClients.ClientNames
FROM tblClients INNER JOIN tblLogins ON
tblLogins.ClientID = tblClients.ClientID
WHERE tblLogin.LoginDate <= #dateadd("m",-
1,now())#
```

If you look at the data from this query, you might see a lot of repeated records, as it's quite likely that clients have logged on more than once in the past month. In this situation (as you are interested only in which clients they were, not when they logged in) you need to use the **DISTINCT** word, to just show single records. The above query would now read:

```
SELECT DISTINCT tblClients.ClientNames,
FROM tblClients INNER JOIN tblLogins ON
tblLogins.ClientID = tblClients.ClientID
WHERE tblLogin.LoginDate <= #dateadd("m",-
1,now())#
```

However, be careful in using **DISTINCT** when you're retrieving a "memo" or "image" type field – it won't work.

How about counting the number of logins in the month? Simple. We can group similar data together and count the items:

```
SELECT tblClients.ClientNames, count(tblLo-
gin.LoginDate)
FROM tblClients INNER JOIN tblLogins ON
tblLogins.ClientID = tblClients.ClientID
WHERE tblLogin.LoginDate <= #dateadd("m",-
1,now())#
GROUP BY tblClients.ClientNames
```

As you can see, queries can start to get complicated – especially when you're involving lots of tables with many conditions. But as they're the means by which data is extracted, getting to grips with SQL can be an important part of a developer's job.

A couple more examples now with slightly more unusual queries:

d. How do I generate a list of both current staff and clients in one datasource? This data is in two separate tables and really isn't connected; we just want one unified list. The solution is a **UNION** query that effectively joins two queries into one dataset:

```
SELECT tblStaffStaffName AS fullname
FROM tblStaff
WHERE leavedate is null
UNION
SELECT tblClients.ClientName AS fullname
FROM tblClients
WHERE leavedate is null
```

This produces one dataset with a single column "fullname" that is the combined list of StaffName and ClientName.

e. Select staff who earn more than the average wage for the company, in descending order.

```
SELECT tblStaff.StaffName
FROM tblStaff
WHERE tblStaff.salary >
(SELECT avg(tblStaff.salary) FROM tblStaff)
ORDER BY tblStaff.Salary DESC
```

This is actually a type of "subquery" and is very useful when you need to nest one query inside another; that is, the main data you need relies on a separate query.

How Do I Refer to the Data Now That I've Retrieved it?

The standard and most obvious way of putting query data onto a Web page is to use the **<CFOUTPUT>** tag and tell it the name of the query. The following example simply lists the data in our query example 2.f (which we might have called "getClientLogin").

```
<cfoutput query="getClientLogin">
#ClientName#, <br>
</cfoutput>
```

The **CFOUTPUT** loops through each record in the query and prints out whichever column you specify. **CFOUTPUT** can be a bit limiting, however, as you're restricted in the particular CF tags you can use inside it. If you've ever had the frustrating message saying that a particular tag is not allowed within a **CFOUTPUT**, you'll appreciate this little foible.

However, the **CFLOOP** tag achieves the same result and you can use whatever tags you want inside. Just be careful to enclose any column names or variables within a **CFOUTPUT** tag pair.

The problem of how to refer to a totally separate query that is different from the one specified in the **CFOUTPUT** tag also comes up. If you merely put the column name in itself, you'll be told that the column referred to wasn't found in the query. The other way is to put the query name in front of the column as follows:

```
<CFOUTPUT query = "getClientName">
#ClientName# :<cfif #ClientName# eq #get-
StarClient.ClientName#>
STAR CLIENT! </cfif>
</cfoutput>
```

By using **#getStarClient.ClientName#** you're specifically telling CF to use the ClientName column from the **FIRST** record in the **getStarClient** query (presumably this query retrieves one record only so we're okay). You're also able to get at specific records in a query by using the syntax

#queryname.columnname[n]#, where "n" is the number of the record you need. If you combine this with the "CurrentRow" property, you can match records and provide a way to overcome the **DISTINCT** problem with "memo" fields.

For example, suppose each client has a memo field in their record to record notes. The query to retrieve their distinct accesses will not work if we want their notes as well. You can get either their notes or their accesses, but not both! Well, how about doing two separate queries then? Fine, but

***"SQL and CF
manipulate the
data, HTML with
some CF tags
presents the
information and
JavaScript does
various 'jobs' in
the background
as required."***

how do you match them up when you come to list the data? You need to use the above **#queryname.columnname[n]#** idea. By making sure that the records are in exactly the same order, we can match them up as in Listing 1.

Moving Data Between Pages

The two normal ways to pass data from one page to the next are via the URL that calls the new page or by using forms. This results in variables being available in the new page as either **URL.variable** or **Form.variable**. CF, however, is flexible about having to say what kind of variable you want to use. If you consider a general form used to edit client data, this could be called by clicking on a client's name (on an anchor tag) or by choosing the client from a listbox on a form. These would produce variables in the new page called "URL."

clientID" (clicking the hyperlink method) or "Form.ClientID". However, if we have to state on the page itself what type of variable it is to expect, we restrict the page to being called by only one method. If we just refer to "ClientID" in the page, it will accept a Form OR a URL variable, which means we can use a single page and don't need two versions.

The "hidden" form field is also useful for passing static data between pages. If you're writing a wizard that takes the user through maybe four sets of choices before the final decision is made, then you need to pass the results of the first three forms along to the fourth. Simply set the values of a "hidden" field each time and this will be passed along to the next form in sequence.

What Else Can I Do with the Data? *Dumping Data into Spreadsheets*

There's often a need to analyze data further in a spreadsheet application, which of course entails transferring the data from the static page into a spreadsheet-readable form. The tag to use in this case is CFCONTENT, a tag that can output data in a number of formats, the most useful of which is perhaps the "tab-delimited" form, where data has tabs between cells and new lines after each line of data. There are alternatives: commas, for example, can be used as data separators to produce so-called "CSV" files (Comma Separated Value files).

Taking the tab-delimited format as an example, however, Listing 2 outputs data into a table on-screen and at the same time builds up a variable that can then be sent to the user's spreadsheet using CFCONTENT.

In Listing 2 the CFCONTENT will stream out the data in the tab-delimited form. Depending on how the user's browser is set up, it can either be saved to the user's hard disk or a spreadsheet application will open automatically for the file to be loaded into.

Updating Separate Tables

You might need to use your retrieved data to update different tables or to append as new records. In this case CF is being used to maintain your databases. Similar to retrieving data, this is achieved by using other SQL statements, namely INSERT and UPDATE. To insert a new record use the following syntax:

```
INSERT INTO tblClients (ClientName,
email, TransactionLimit)
VALUES ('M Smith', 'msmith@somewhere.co.uk',
800)
```

The UPDATE statement uses the following syntax:

```
UPDATE tblClients
```

```
SET email = 'rcarter@anywhere.com',
    Tel = '+44 181 375 0446',
    TransactionLimit =
600,
    Contractdate = '1998-05-02'
WHERE clientID = 346
```

Notice the use of quotes around the e-mail, Tel and Contractdate fields. In general, the rule is to use quotes when referencing a text or date field, but not numbers.

Some Extra JavaScript Tricks

JavaScript is a quite powerful programming script that can be used to do a variety of background jobs, including interacting directly with the user (through "alert," "confirm," etc.) and redirecting pages. However, developers need to be careful in using JavaScript as users can disable it in their browser.

A particular use of JavaScript is for validating forms before they're submitted. This is in fact what those CFFORM elements do. If you look at the source of a Web page with a CFFORM, you can see that spurious JavaScript has mysteriously appeared at the top of the page.

With a little knowledge of how JavaScript operates, you can implement some simple checking that happens on the client side. The beneficial effect of this is that the form doesn't need to be submitted before it's checked and is therefore much quicker than checking in the action template.

The procedure for checking a form can be done in two ways: 1. either as a statement in the FORM tag (or CFFORM tag) saying 'OnSubmit = "somefunction()"' (which means that as soon as the form is submitted a JavaScript function that has been written into the header of the document is run) or 2. it can be tied to the submit button itself using the "OnClick=" procedure. Listing 3 shows how you can make sure that a textbox called "newdata" in a form called "editform" has at least some text in it.

The function checks to see if the textbox is in fact empty. If it is, a message pops up to say, "You must submit a value..." and FALSE is sent back to the form, meaning it isn't allowed to be submitted. If the textbox has been filled in, however, TRUE is returned and the form can be submitted.

JavaScript can do many other tricks. A useful piece of code to know is "this.document.location.href=", which is the JavaScript equivalent of <CFLOCATION>. It redirects the current document to the location specified. ColdFusion variables can also be used in this context so the following piece of code will redirect another frame's reference, passing on a variable to be used within that page:

```
<head>
<cfoutput>
<script language="JavaScript">
function resetresult(){
    top.result.document.location.href =
'newpage.cfm?ClientID=#ID#'
}
</script>
</cfoutput>
```

Of course, there's no reason why that variable couldn't have been passed into the function itself when it was called, so this is functionally equivalent:

```
<head>
<cfoutput>
<script language="JavaScript">
function resetresult(IDNum){
    top.result.document.location.href =
'newpage.cfm?ClientID=' + IDNum
}
</script>
</cfoutput>

<a href="dummy.cfm" OnClick="<cfoutput>
resetresult(#IDNum#)</cfoutput>; return
false">Edit Client</a>
```

Notice the "return false" statement. This prevents the anchor tag from actually jumping to the new page, so all that happens when you click the hyperlink is that the function is called and the CF variable is passed into it. The CFOUTPUT tags are nested inside the anchor tag to make sure that the value of IDNum is passed, not just the text itself!

A nice touch in JavaScript is the ability to open and have control of new browser windows. This can be useful for showing intermediate edit or search forms. The following JavaScript function will open a new window and, given the URL, height and width parameters, size it and put the requested URL into it. Also, if the window has previously been opened and is still opened, it will bring the window on top of any other windows and then load the new URL into it.

```
<HEAD>
<script language="JavaScript">
var hWinnd=null
function newwin(href, winwidth, winheight){
    if (hWinnd == null
|| hWinnd.closed){ hWinnd =
window.open(href, 'menuwindow', 'width='+winwidth+', height='+winheight);
    hWinnd.focus();
    } else {
    hWinnd.focus()
    }
}
</script>
</head>
```

When this new window has performed its appointed task, JavaScript can close it automatically using "this.close()"; you can do this only with windows that JavaScript has opened in the first place.

However, there are a couple of things that you should remember: JavaScript is case-sensitive. Be careful how you name elements in your forms. If you call a textbox "Edit1", JavaScript will tell you that "edit1" doesn't have any properties, and that's true because it was looking for "Edit1". JavaScript can also be placed anywhere within the page, so if you want to change the status bar at the bottom of the browser whenever the mouse goes over a nice image, you can use the following syntax:

```
<img source="prettypic.jpg"
OnMouseover="window.status=' Nice Picture!'"
OnMouseOut='(window.status='') ">
```

It's also possible to actually set the values of form elements before submission. For instance, when a hyperlink is clicked, a JavaScript can run to set a hidden form variable to a value to record what has happened.


Conclusion

We've skimmed the surface of some elementary parts of CF, SQL and JavaScript. It's important to remember, however, that things rarely work out the first time. Many developers adopt a "try it and then correct the errors" principle, and there's often an alternative (possibly less complicated) way of achieving the stated objective.

There are many exotic and underused tags and functions in CF that perform useful tasks, but it's possible to write very good, effective code without in-depth knowledge.

Learn what's appropriate to your application; as you progress you'll learn a whole

lot more. Never be afraid to try something new or unfamiliar (just be careful not to crash the server!).

SQL and CF manipulate the data, HTML with some CF tags presents the information and JavaScript does various "jobs" in the background as required. Mixing a good dose of all of these, in proportions tailored to a user's particular needs, produces effective, efficient pages that hopefully will work seamlessly together. 

About the Author

Andy Birchall has been both a company accountant and a ColdFusion developer for MediaTel Ltd. for the past two years. He learned ColdFusion from scratch and has mainly been involved in rewriting the existing internal management/contacts system in ColdFusion, as well as writing code for MediaTel clients. He can be reached at xfd47@dial.pipex.com.

xfd47@dial.pipex.com

Listing 1.

```
<cfquery name="getAccesses" datasource="clients">
SELECT DISTINCT tblClients.ClientID, tblClients.ClientName
FROM tblClients INNER JOIN tblAccess ON tblAccess.ClientID =
tblClients.ClientID
WHERE tblAccess.AccessDate < #dateadd("m",-1,now())#
ORDER BY tblClients.ClientID
</cfquery>

<cfquery name="getNotes" datasource="clients">
SELECT tblClients.Notes, tblClients.ClientID
FROM tblClients
WHERE tblClients.ClientID IN (#valuelist(getAccesses.ClientID)#)
ORDER BY tblClients.ClientID
</cfquery>

<cfoutput query = "getAccesses">
#ClientID# #ClientName# #getNotes.Notes[CurrentRow]#
</cfoutput>
```

Listing 2.

```
<!-- set up dummy variable called "taboutput" to hold the tab-
delimited data -->
<cfset taboutput = ''>
<!-- set the tab charactes and carriage return variables -->
<cfset newln = chr(13)>
<cfset tab = chr(9)>

<!-- set the header line for the spreadsheet table -->
<cfset taboutput = taboutput & "Name" & #tab# & "Address" & #tab#
& "Telephone" & #tab# & "Email" & #newln#>

<!-- the query "qryClientDetails" picks up Client s name, address,
tel and email -->

<!-- normal HTML table to display the data -->

<table>
<tr>
<th>Name</th><th>Address</th><th>Telephone</th><th>Email</th>
</tr>
<cfoutput query="qryClientDetails">
<tr>
<cfset taboutput = taboutput & #ClientName# & #tab# & #Address# &
```

```
#tab# & #Telephone# & #tab# & #email# & #newln#>
<td>#ClientName#</td><td>#Address#</td><td>#Telephone#</td><td>#ema
il#</td>
</tr>
</cfquery>
</table>

<!-- output the data into tab-delimited file -->
<cfcontent type="text/tabdelimited"><cfoutput>#taboutput#</cfout-
put>
```

Listing 3.

```
<HEAD>
<TITLE></TITLE>
<script language="JavaScript">

function checkform() {

    if(this.editform.newdata.value==''){
        alert('You must submit a value to edit in this textbox')
        return false;
    }else{
        return true
    }
}
</script>
</HEAD>

<BODY>
<FORM name="editform" action="EditPage.cfm" method="post">
.
.
.
<input type="text" name="newdata" >
.
<input type="submit" value="Submit form" OnClick="return check-
form()">
</form>
</BODY>
```

CODE LISTING

The complete code listing for this article can be located at www.ColdFusionDevelopersJournal.com



Creating a 'Live' ColdFusion Site

Requesting information from the server without user intervention

by Raymond Camden

Normal Web browsing is a stateless affair. This means that until you request something from the server, your browser sits idle, like a lump of clay. Ignoring Java-based apps or chat clients, most Internet browsing happens on a very simple basis. You (via your browser) ask the server for a file and the server complies by spitting a page back at you. State can be maintained by the use of cookies or special variables on the URL string, but information still comes only when you request it.

What I'll demonstrate in this article (and the ones that follow) is a simple way to create a "live" Web site – one that will request information from the server, and vice versa, without user intervention. This brings a whole new level of interactivity to the browser without forcing the user to download a separate plug-in or application, or resort to high-overhead Java applets.

How can we use this in the real world? Imagine a Web site that reflects polling information live on the page. Consider a game of checkers designed with DHTML that allows you to play against anyone in the world. What if you were notified when clients logged on to the Web-based project management tool you developed?

Creating the Framework

To begin, let's create a very simple frames-based page. The example in Listing 1 creates three frames. The first is the menu frame, which would normally contain links to major parts of the Web site or Web application. The third frame points to main.cfm. Again, this would normally be the front page of a site or application. For the purpose of this demonstration, you may wish to create menu.cfm and main.cfm as files on your system so that you don't get constant error messages. You may fill them with anything since they don't relate to our example.

The second frame, however, is special.

First of all, we don't point it to a URL. Instead, we use a little JavaScript trick to create the source for the frame. We tell the frame to use a JavaScript variable called *blank*. Notice that since we are inside the frame, we must use *top.blank* to refer to the variable. The variable is set to a simple BODY statement. Even though we've specified that the frame should be zero pixels wide, the browser always shows a little bit of the frame. (This is essentially a security issue. If a frame could be truly invisible, an evil Webmaster could use it to snoop on you while you surf the Web.) By setting the source of the frame to a simple color, it appears to be part of the design of the site instead of an error in the HTML. The size of this frame will differ and you should be aware that in MSIE 5.0 it doesn't appear at all.

JavaScript to ColdFusion Communication

Now that we have our frames, it's time to set up the first part of our live site. We need to find a way for the browser to send messages to the server without the user's intervention. This is where our hidden frame comes in handy. First, let's set up an event to happen every 10 seconds. We do this with the JavaScript *setTimeout* function, which allows you to call another function after a specified number of milliseconds. If that function calls *setTimeout* again, you essentially have a function that will run indefinitely. (There is a newer function called *setInterval* that creates the same effect without the need to constantly call itself again. Since it's available only in 4.0 browsers we won't use it in this example.) We initialize our looping function with the *onLoad* event handler in the FRAMESET tag:

```
<FRAMESET COLS="150, 0, *" BORDER=NO FRAMEBOR-
DER=0 onLoad="heartBeat()">
```

That statement tells the browser to call the function 'heartBeat' immediately after loading. To create the loop, the heartBeat function must use the *setTimeout* function to call itself.

```
function heartBeat() {
    setTimeout('heartBeat()', 10*1000);
}
```

Notice that I used 10*1000 to represent 10 seconds. I could have just written 10000, but this helps me to remember that the function uses milliseconds. To communicate with the server, we'll reload the hidden frame each time the heartBeat function is called.

```
function heartBeat() {
    top.Server.location.href = "server.cfm";
    setTimeout('heartBeat()', 10*1000);
}
```

Our function has been modified, so every 10 seconds the hidden frame, whose name is *Server*, will be sent to a specific page called *server.cfm*. The whole point of this example is to show communication between the browser and the server, so let's send something interesting to the server, in this case our browser information.

```
function heartBeat() {
    var serverURL = "server.cfm";
    top.Server.location.href = serverURL +
    "?browser="+escape(navigator.userAgent);
    setTimeout('heartBeat()', 10*1000);
}
```

There are a few things to note here. First, we broke off our *server.cfm* location into a variable so that it would be easier to modify. Then, instead of just moving our hidden frame to the new location, we appended information to the query string, a list of information that is passed along with the URL. Our ColdFusion file will be able to access the browser information simply by checking *#URL.browser#*.

ColdFusion to JavaScript Communication

Now that our Web page is communicating with the server, we need to make sure

it's listening. Here's the source code for my first version of server.cfm:

```
<!-- Set up initial values -->
<CFSET BROWSER_INFO = URL.browser>
<CFSET CURRENT_TIME = DateFormat(Now(), "mmmm
d, yyyy") & " " & TimeFormat(Now(), "h:mm:ss
tt")>
<CFSET FILE = ExpandPath("log.txt")>

<CFFILE ACTION="Append" FILE="#FILE#" OUT-
PUT="#CURRENT_TIME# = #BROWSER_INFO#">
```

What does our simple little CFM file do? Nothing much, right now. First we grab the information that the JavaScript sent us in URL.browser. Next we create a string that contains the current date and time. This will be used as a time stamp so we know when the information was sent. We create a reference to a file in the same directory as an application, and finally we save our information by appending it to the file. You may wish to try this out now. Launch a copy of Netscape and MSIE and just sit back. After a minute or so, open up log.txt and examine the lines. You will probably see something like this:

```
August 16, 1998 2:10:26 AM = Mozilla/4.0
(compatible; MSIE 5.0b1; Windows 95)
August 16, 1998 2:10:35 AM = Mozilla/4.5b1
[en] (Win95; I)
```

```
August 16, 1998 2:10:36 AM = Mozilla/4.0
(compatible; MSIE 5.0b1; Windows 95)
August 16, 1998 2:10:46 AM = Mozilla/4.0
(compatible; MSIE 5.0b1; Windows 95)
August 16, 1998 2:10:55 AM = Mozilla/4.5b1
[en] (Win95; I)
August 16, 1998 2:10:56 AM = Mozilla/4.0
(compatible; MSIE 5.0b1; Windows 95)
August 16, 1998 2:11:05 AM = Mozilla/4.5b1
[en] (Win95; I)
```

There are a few interesting things to note here as well. We are essentially writing a file to the server with JavaScript, something that is normally impossible. This technique could be used to save form data or other information without user intervention. Also, each time a browser hit is recorded, it is either 10 or 20 seconds after the last hit, and you can't guarantee that the heartBeat function will work at every interval. In this case the user of Netscape 4.5 may have hit the back button on his or her browser and then returned to the demo application. Whatever the reason, like other applications that set up a net connection, don't count on every "ping" happening at a set time.

At this point our ColdFusion isn't doing anything terribly interesting. We need to set up the communication lines by using the ColdFusion file to return data to the JavaScript function. But what exactly should

we return? Well, if we were saving information that was a bit more interesting we might want to know what information has been saved since the last time we pinged the server. In this case the ColdFusion file needs to know the last time we were here. Let's assume that our JavaScript will send this variable along with the browser information by way of URL variable. We need to go back to our JavaScript function and add that functionality. We'll add a variable called LastTime, which will be blank at first. After sending the request to the server, we set LastTime equal to now so that on the next call it will represent the time 10 seconds ago.

Also, we want the variable to persist, so we declare it outside the function. Since we want to create a pretty time stamp, as we did in the ColdFusion file, we will create a function outside heartBeat to return this string for us. We also need to return it so ColdFusion can use it in a DateCompare function. To simplify this we'll create a Day/Time string in the same format as our ColdFusion file, shown in Listing 2.

The prettyDate function is pretty simple, except for a few hacks to get around the fact that the getYear function returns a two-digit number for years in this century and a four-digit number for any other year. Also, since the getHours function works on a 24-hour clock, we must convert it to nor-

RSW Software

www.rswsoftware.com

mal time to see if we need to switch AM to PM. Notice that when heartBeat is called for the first time, lastTime will be null. This is fine. We'll make our ColdFusion file assume that if lastTime is null, the browser needs to send back all information, not just information since a specific time. After every other ping on the server, lastTime will be defined and the server will send back newer files only.

Now that we're sending the proper information to the ColdFusion file, let's direct the communication back to the JS file. Listing 3 contains the edited server.cfm file.

Again, it's best if we take this step by step because this is a pretty big addition. First, we set a variable called LastTime that is set by the URL variable. As I wrote earlier, this will be blank at first. We also add two variables that will help us later on. One is NL, which represents a new line character and will help us read the text file. The second is a loop counter called X. This is necessary to set the index values of our array.

Our next change is to read in the file we just appended to, into a variable called BUFFER. After we've read in the file, we need to prepare to send the information back to the JavaScript. Remember that this is a hidden frame, so we never need to output any real HTML. Instead, we immediately begin by outputting a <SCRIPT> tag, followed by

the declaration of a new array called bufferArr. There are two things to notice here. First, we did not create this array in our HTML file because we'll need to re-create it for every ping on the server. Since the array will represent "new" data for us, we have to be sure to create a new instance of it. Second, notice the use of top.bufferArr. Don't forget that we are within a frame of a larger frameset. To talk to the page/application as a whole, we must use top to direct our data.

Depending on whether or not we have hit the server, the variable LAST_TIME will be set to a valid date or it will be blank. If it's blank, we want to return every line of our file. We use a simple CFIF statement block to create a date object out of the LAST_TIME variable. If it's blank, we create a Date object that we know will be before any lines of the file. If it's not blank, the Date object represents the last time we hit the server.

Now our loop begins. Since we set the NL variable to represent a new line, we use that as our delimiter for the BUFFER variable. This allows us to process each line of our text file by using the variable we declare in INDEX, CURR_LINE. We know that each line of our file follows a certain syntax – in this case `TIMESTAMP = BROWSER INFO`. With this in mind we can do a bit of fancy string manipulation to get the `TIMESTAMP` out of the line. By pretending that the line is a list

in itself, we first use the `ListGetAt` function to grab the first element and be sure to let it know that the delimiter is an equal sign. Since this will return our `TIMESTAMP` with an extra space at the end, we also use `Trim` on the returned value. We use the same trick to grab the browser information out of the string. Once we have a nice string representing the `TIMESTAMP`, we create a date object out of it. If this particular line has a `TIMESTAMP` that is after our `LAST_TIME` variable, we have new information. We increment our `X` value and set the value of this array index to `THIS_BROWSER`. At the end of our loop we must spit out a `</SCRIPT>` tag so the browser knows we're finished. To process our date, we call a function (`top.displayBuffer`) that doesn't exist yet. The next section will detail how we handle this data.

Displaying the Server's Information

Our "live" application is almost complete. We have the browser sending constant requests to the server and now we have our server sending information back. But what do we do with this information? For the purpose of this article we'll simply dump it to the screen. Since we have an array full of data, and the ColdFusion file calling a function called `displayBuffer`, it would probably be a good idea to actually code this function:

Igneous

www.?????.com

Advertiser Index		
Advertiser		Page
Able Solutions	360-253-4142	2
Allaire Corporation	888-939-2545	3, 17, 21, 51
Catouzer Inc.	604-662-7551	4
Eprise Corporation	800-274-2814	9
FusionFX	800-780-2422	46
Infoboard	800-514-2297	33
Intermedia.net	650-424-9935	52
Live Software	408-996-0300	15
RSW Software	508-435-8000	27
Schlumberger	800-825-1155	13
SYS-CON Interactive	800-513-7111	34,35
The Igneous Group	877-469-ROCK	28
Virtualscape	212-460-8406	29

```
function displayBuffer() {
    top.Main.document.open();
    for(i=0; i<bufferArr.length; i++) {

top.Main.document.writeln(bufferArr[i]+"<BR>
");
    }
    top.Main.document.close();
}
```

This function opens the document object of the main frame and dumps the contents of bufferArr to it. We also append a
 tag to each line so it's easier to read each one. The first time you run this application, you should see every line of log.txt dumped to the right frame. After that, you'll see only one line, the line representing the

"ping" you just sent to the server. To make things truly interesting, launch your other browser, preferably MSIE if you're using Netscape, and notice how it also begins to appear in your frame. What's neat about this is that the MSIE browser you just pointed to the application could actually be someone else's Web browser. Next, imagine four or five other people hitting your application at the same time. Suddenly you're seeing their impact as the data is displayed on your screen. Welcome to your first "live," albeit boring, Web application!

Concluding Thoughts

The ideas in this article are only the beginning. We designed a living Web page (with a heartbeat!) that stays in constant

communication with the server. Other visitors to your Web site open up their own lines of communication and, in turn, everyone gains awareness of the other users. In the next article we'll dive deeper into the concepts presented here and begin to work on our Web-based chat server. [CFDJ](#)

About the Author

Raymond Camden is a contractor/consultant based in the Silicon Valley and has developed with ColdFusion for two years, including online commerce sites, project and document management tools, numerous Custom Tags, and the infamous Death Clock. He helped form the Bay Area ColdFusion User Group and is a contributing author to CF Advisor.

morpheus@deathclock.com

Listing 1.

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Sample Application - Level 1</TITLE>
```

```
<SCRIPT>
```

```
var blank = "<BODY BGCOLOR=' #FFFFFF' ">";
```

```
</SCRIPT>
```

```
<FRAMESET COLS="150,0,*" BORDER=NO FRAMEBORDER=0>
```

```
<FRAME SRC="menu.cfm" NAME="Menu">
```

```
<FRAME SRC="javascript:top.blank" SCROLLING=NO NAME="Server">
```

```
<FRAME SRC="main.cfm" NAME="Main">
```

```
</FRAMESET>
```

```
</HEAD>
```

```
<BODY>
```

```
Upgrade to a browser made this century, bozo.
```

```
</BODY>
```

```
</HTML>
```

CODE LISTING

The complete code listing for this article can be located at www.ColdFusionDevelopersJournal.com

Virtual Scape

www.virtualscape.com



Transitioning to ColdFusion from CGI Programming

ColdFusion and the Web go hand in hand to an era of more sophisticated applications

by Jim Esten

There's little disputing the pace at which the Web has evolved over the past several years. Fueled by graphical interfaces to a suite of early text-based protocols, the leap into the Internet marketplace by providers of previously proprietary content systems and the plunge in street prices of personal computers, growth of the Internet – most notably the World Wide Web – continues at a torrid pace. In conjunction with the sheer growth has been the evolution of the Web from a place to browse information to a real infrastructure for the development of sophisticated applications.

Five years ago I spent my time on Web servers installing Perl scripts written to the requirements of the Common Gateway Interface (CGI). Soon after, I began my own journey into the world of Perl programming, first as a UNIX system administrator and later as a Web developer. CGI was easy to understand, easy to set up, easy to implement. Then along came Web servers that ran on something other than UNIX platforms. CGI still worked. It needed a tweak here and there and I now had to keep track of win-cgi and dos-cgi in addition to the stuff I already knew, but at the heart CGI was still pretty easy to work with.

About three years ago the whole Web seemed to come into full bloom. Those were great days for CGI programmers (yes, programmers!). New applications appeared daily; the free software movement created a rich environment for coders to learn from each other's work, and the browsing community started asking for more. More dynamic elements, more access to things elsewhere on the Web and, most of all, access to things like corporate databases. It was about this time I began a personal crusade extolling the virtues of Web applications. They were relatively quick to develop. They were easy to deploy to remote locations. They gave you a single point of

maintenance. Slowly, people listened. CGI applications flourished.

There was a problem, though. For all the wonderful things CGI applications could (and still can) do, they really did require programmers to construct them and people rather familiar with the operation of Web servers to set them up. To this day, CGI programs still carry the somewhat deserved reputation of being inefficient because of the way Web servers handle the requests. The landscape was now mature enough for a serious Web development environment. Allaire's release of ColdFusion took the development community by storm. The ability to embed a rich set of programming constructs in a markup language – and further integrate that markup language directly in standard Hypertext Markup Language pages – gained immediate attention.

This integration of programming constructs within a markup language is perhaps the fundamental difference between traditional CGI programming and inline programming from the point of view of the application developer. When we move to an environment like ColdFusion, we must think differently about how to architect an application.

Let's look at a quick sample of each in action doing a very common task. It's almost painfully common to see the date and time displayed on a Web page. I say "painfully," because if you think about it most computer users probably already have the date, the time or both already displayed somewhere on their desktop. I swap routinely between sessions on Solaris (date and time displayed in the CDE toolbar), Windows NT (time displayed in the lower-right corner of the task bar, date viewed with a mouseover) and Mac OS/8 (time next to the Finder icon). Why this duplication of effort has become so common can only be attributed to the ease with which it is done

and the ability to show the time at the server instead of the client.

First, the CGI way. This actually requires a combination of tools. Very often a CGI script is called via a server-side include "exec cgi" tag. Here's the script (UNIX style):

```
#!/usr/bin/perl

($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst) =
    localtime(time);
print "The time is $hour:$min:$sec" ;
```

This is called from the Web page with something like:

```
<!--#exec cgi="/cgi-bin/thetime.pl" -->
```

This produces the result shown in Figure 1.

This method works, though the different configurations for server-side includes may make it something less than completely portable. Experienced CGI programmers will note the absence of a content header. I've run into Web server configurations that require it as well as those that don't.

Now let's put that same thing into our home.cfm page and see what changes.

```
<CFOUTPUT>
The time is #TimeFormat(Now(), "HH:mm:ss")#
</CFOUTPUT>
```

This results in exactly the same output format as shown in Figure 2.

Two points are significant here. First, the CGI implementation requires that server-side includes be enabled. Second, the CGI implementation uses a separate script, which must be installed, configured and permissioned correctly to work. Even Perl bigots like me can be swayed by the ease of the task with ColdFusion.

This example illustrates a difference in the architecture between CGI-driven applications and those constructed with inline scripting tools. At the very heart of the difference is the separation of program code from display code. Under the "standard" CGI configuration on Web servers, all files

intended to be run as scripts must reside in a specific directory tree. Servers are often configured to map this real system directory to the URL "/cgi-bin." The display files are most often separate HTML files that reside in the document tree of the Web server. Scripts must be configured to be executable by the user ID under which the Web server runs. The installation and setup of even a simple feedback form will require managing files in two separate directories.

A typical ColdFusion application will place all files in the same directory somewhere in the Web document tree. The "display" and the "processing" may well take place in separate files, but the front end of this is very likely to have dynamic elements such as retrieving information from a database to populate a select menu. To accomplish that process in a CGI application would require the script itself to generate the HTML. I have commonly deployed CGI programs in this manner, but have found them very inflexible. Any changes to the visual interface must be made to the script itself – often by a nonprogrammer whose comfort level reading a programming language leaves a bit to be desired. To further complicate matters, the editing is usually done locally (note that most of my CGI travels have been on UNIX systems) and files are then uploaded. As often as not, uploaded as binary and corrupted. A one-liner in Perl fixes that problem faster than I could probably teach a graphic artist how to get around in VI, so I've at least learned to take that one in stride!

So what about more useful things? In the old days on the Web (you know, way back in about 1995), pretty much everything was accessible to anyone who was browsing. Not so anymore. These days I find myself logging into all kinds of things. And the site is keeping track of me somehow, often in a database.

Databases come in various shapes, sizes and degrees of scalability. From simple flat text files to large-scale data warehousing systems, there are database solutions for storing all types of information. Because ColdFusion is platform-specific, running only on Win32 and Solaris, I'll confine the following example to a readily accessible product, Microsoft Access running on a Win32 system. I typically develop on Windows NT v. 4.0 and find it much easier to move things to UNIX from Windows than vice versa.

To implement a login, several things occur. The user is presented with a login dialog box with inputs for a user name and usually a password. The user clicks a submit button and the form is processed by an authentication script that connects to a database, looks up the user name and, if

found, compares the stored password with the input. A successful match gets the user to the desired page. A login failure does something else, hopefully something intuitive enough to the user to determine the next course of action. Simple enough concept, and accessible enough to show the ease of transitioning from traditional CGI to inline scripting.

Figure 3 shows a simple login screen. The form fields are "user" and "pass," which for the sake of clarity we'll assume exist in the database as columns username and password. Listings 1 and 2 contain the complete source to the login screens. I'll parse them into variables whose names match the form elements for the CGI example. We'll query the database with the user input, then examine the password field from the query and compare it with the pass input. For the CGI implementation I'll use Dave Roth's Win32::ODBC.pm module.

The complete source can be found in Listing 3. We'll show just the pieces relevant to the database query here. Note that, for simplicity, the examples are shown without any error-checking routines. This would obviously be poor practice in a production environment!

First connect to the database (using a previously defined System DSN):

```
Sdb = new Win32::ODBC("members");
```

Then do the query and load it up into a hash:

```
Sdb->Sql("select * from logins where user = $user");
Sdb->FetchRow();
%matches = Sdb->DataHash;
```

At this point, if the query was successful, we should have a single record stored in an associative array (a hash, in contemporary PerlSpeak), which we can work with in the same manner as the parsed form data (see complete code listing).

To do my comparison:

```
if ($pass eq $matches{'password'}) {
    print "Location: Surl\n\n";
} else {
    print "your login failure here\n";
}
```

This is a bit simplistic, but the basic



Figure 1: Generating the current time via a server side include tag

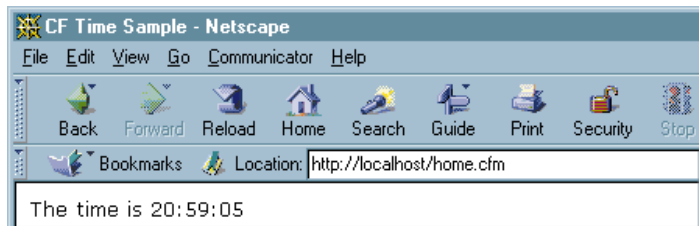


Figure 2: Generating the current time with ColdFusion

steps are:

1. Set up the module for use.
2. Create an instance of it.
3. Query the database.
4. Load the returned data into a data structure.
5. Do comparisons between stored data and user inputs.
6. Direct to appropriate page.

Now let's take a look at an equivalent sequence with ColdFusion. Using a similar front end (Listing 2), I then do the query (Listing 4) as follows:

```
<CFQUERY NAME=" matches" DATASOURCE="
members" >
    select * from logins where user = \quote
#user#
</CFQUERY>
```

Again, perform the comparison:

```
<CFOUTPUT QUERY=" matches" >
<CFIF #password# IS #pass#>
    <CFLOCATION URL = "success.html" >
</CFELSE>
    <CFLOCATION URL = "failure.html" >
</CFIF>
</CFOUTPUT>
```

The sheer amount of code required to do the database-specific functions is roughly equivalent. Not apparent is the amount of work required to get to this point. The form must be parsed for the CGI example. ColdFusion handles this internally. The error handling shown in the full code listing requires some handling differences, though these can be minimized by using a global error routine. I often place this error routine in an external file and include it in my CGI script. In ColdFusion a similar approach is to place shared (global) items in the APPLICATION.CFM template.

Project-based tools have become stan-

Welcome! Please type your username and password below and click the *Login* button to continue.

Username	jesten
Password	*****

Login Clear

Figure 3: A simple login screen

dard fare in the application development world and have made their way into the Web application world as well. Comparing development environments between traditional CGI programming in Perl and ColdFusion is a bit of a lopsided battle. While several products have recently emerged with visually attractive interfaces for programming in Perl, I've yet to find a competent product that is effective for CGI programming. I personally rely on a marvelous programmer's editor called UltraEdit, which has language-specific dictionaries to do basic syntax highlighting and has configurable menus that let me call up Perl's built-in debugger to run syntax checks on scripts in a separate window. From my current menu setup, shown in Figure 4, I can either run a script or simply run the Perl interpreter in syntax check mode. Both produce output in the editor's own results window.

Great product at a great price, but as good as it is, a text editor is hardly a substitute for a full-featured development environment such as the ColdFusion Studio.

Like its little brother HomeSite, the ColdFusion Studio serves the dual function of a project organizer and graphical coding environment. Though I tend not to use the individual tag shortcut buttons, having cut my markup language teeth on VI, things like tag completion, context-sensitive tag help and wizards to set up things like framesets and table are wonderful productivity boosters. The CF Studio's project manager falls well short of a version control product, but does provide a convenient means of keeping files organized (see Figure 5).

Since the standard CF-driven application will often reside in a single directory, a very

convenient feature is to open the remote documents, edit them and save them back to the remote site. Additionally, making the same change to a collection of documents is easily accomplished using the extended replace command on the entire project. Somebody was thinking!

Debugging is a somewhat quirky thing in markup languages and, after all, the ColdFusion Markup Language is really just a markup language, albeit one that includes standard constructs for conditional programming structures and access to the Standard Query Language. The CF Studio comes integrated with the CSE HTML Validator. It won't debug your SQL statements, but it can at least help you get accurate renderings of display code.

Another area where the CF Studio shines is the application wizards to help programmers get started on the most common types of pages. One in particular, the Verity wizard, is an absolute gem for getting a simple search function running on a Web page. Wizards exist for HTML, Dynamic HTML and ColdFusion pages. While you're unlikely to use a wizard to build a large application, they make great learning tools.

The feature I find most appealing about the CF Studio is the connectivity to the data sources during development. As databases grow, tables multiply as well as get complex themselves. It can be difficult just to remember field and table names. From within the development environment you can view the tables and the columns within the tables of your data sources. This saves me having to keep a database access tool running or keep a database diagram taped to my monitor.

So what happens after you've developed your application and are ready to put it out onto the Web? I've spent a fair amount of time in a shop whose pet term is to "deploy applications." As a former military officer, the thought of deployment conjures up a set of images quite removed from the Web! Whatever you call it, you do need to get your work off your local hard drive and onto a Web server.

To move a CGI-driven application to a production server, you'll typically create a bunch of directories, upload files to those directories, set permissions on directories and files and hope you've got all the right directory paths, URL mappings and access levels.

Deploying a ColdFusion application removes a bit of the rigor. Create a directory in the Web document tree and put all the files in it. Done. Finished. That's the whole process. The first time you do one like this, you'll think twice about how to construct your next application. I can safely call myself a Perl junkie. It's a fun language to work in, and for someone with admittedly modest programming skills, CGI provides some very comfortable coding standards. I'm not likely to leave that world behind, but have found some new life for some existing applications by leveraging the power of a data warehouse behind them.

For those still not swayed to at least take ColdFusion for a test drive, I'll summarize what I feel are the strengths and weaknesses of the two Web application development strategies.

CGI Strengths over ColdFusion

Perhaps the strongest push to use CGI applications is the rich collection of ready-made resources freely available on the Web. Hundreds of free applications can be found, complete with source code. My favorite site for script resources, www.cgi-resources.com, has hundreds of free and low-cost scripts available in all the most popular languages for coding CGI scripts. Allaire has a fairly extensive tag gallery of contributed items, but ColdFusion-driven applications are hard to find, and when I have found them they tended to be a bit expensive.

CGI is largely portable. I've written CGI scripts that run virtually unchanged on multiple versions of UNIX and Win32 platforms. ColdFusion runs only on Win32 and Solaris. A version of ColdFusion running on Linux would probably balance this one.

Development and deployment price is greatly reduced. Perl and several other languages used for CGI programming are free. Most Web servers provide configuration options for CGI support. Using the Apache server running on a Linux-based system with Perl CGI programs, the only cost is the

hardware. The ColdFusion application server is nearly \$900, the CF Studio is almost \$300. Since it runs only on Solaris and Win32, add an expensive (both to buy and maintain) operating system and very robust hardware. CGI wins this battle hands down!

So why switch? Well, I'm not advocating the abandonment of CGI in favor of ColdFusion. Rather, I'm suggesting that this is a mature tool that deserves a concentrated evaluation.

Where ColdFusion Shines

Database connectivity is easier with ColdFusion. Particularly for large commercial SQL servers ColdFusion sets up without much additional software. Version 4.0 features native drivers for Sybase and Oracle. For anyone who has fought the Perl DBD/DBI battles, the ease of connectivity is refreshing.

Deployment is simpler with ColdFusion. Much simpler, in fact, often requiring little more than uploading files and setting up a System DSN to access (and this can actually be done through the Web interface to the ColdFusion Application Server!). CGI applications seem plagued by the permissions issue, which will differ from server to server based on the local configuration. The sites doing ColdFusion hosting at this time are still few and scattered, but the number is growing.

ColdFusion offers a true integrated development environment that rivals that of any popular programming language. This environment ties directly to the remote site out on the Web, making the process of deploying files to a production environment even easier. Table 1 summarizes the discussion above.

A number of typical tasks handled by ColdFusion are of particular interest to CGI programmers. I'll handle some of these in depth at a later date. CGI lives by forms, and forms are made effective through validation of user inputs. Sometimes we enforce our validation rules in our scripts, but this is horribly inefficient. ColdFusion offers us special form element tags that, when sandwiched inside the <CFFORM> tag, allow the ColdFusion application server to generate JavaScript client-side validation routines. The first time I used this, I

	COLDFUSION	CGI PROGRAMMING
Free Source Code		X
Portability		X
Price		X
Database Connectivity	X	
Deployment	X	
Development Environment	X	

Table 1: The comparison table

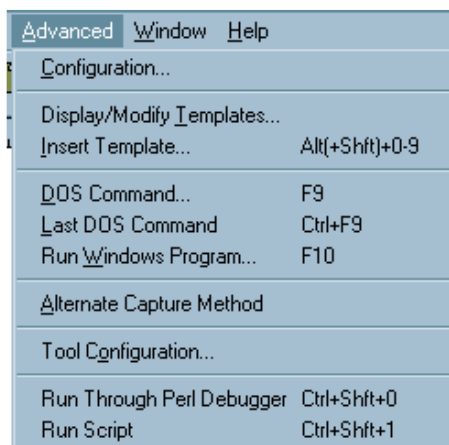


Figure 4: Custom menu options

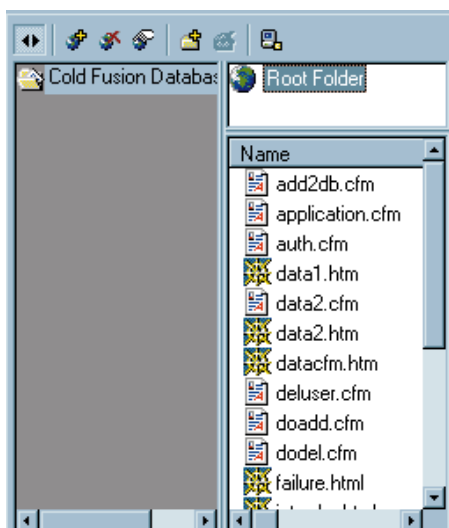



Figure 5: Organizing files as a project

was pleasantly surprised to see very clean, cross-browser-compatible JavaScript that looked amazingly like the routines I so painstakingly write by hand.

Another task I deal with frequently is sending mail. Whether sending order confirmations, announcements or simple feedback forms, I use mail in my CGI applications more and more frequently. Early on this was easy. A system call off to sendmail and messages were on their way. Then came the Win32 servers. No system mail. So I went to socket routines. Very portable, but required a bit of configuration and somewhat tedious to work with. ColdFusion bundles SMTP. The <CFMAIL> tag reads like a simple form. You put your normal mail header parameters with the opening tag and put the content of the message before the closing </CFMAIL>

tag. Simply speaking, it's a thing of beauty. It gets even better since you can specify the results of a query (say, of e-mail addresses) as the "TO" parameter.

Built-in session management features, Java support, the Verity search engine, file and directory management and hypertext transfer protocol interfaces round out the more accessible and useful built-in features. While all these things can be accomplished with CGI, they don't come standard as part of the typical core language distribution. Even with a free language such as Perl, the effort spent configuring the add-on modules weighs heavily in the development time when implementing many of these features and may not be an option at all if you're running your site on a remotely hosted virtual server.

Is CGI going away? Hardly. If you want to add a few simple dynamic elements to your Web site, CGI is a great way to get started. As your needs evolve, especially if they evolve into database programming, ColdFusion is a product you can't afford not to evaluate. Take it for a test drive. Evaluation copies are available from Allaire at www.allaire.com. You can also get an evaluation copy on CD packaged with Ben Forta's *ColdFusion Web Application Construction Kit* – a must-have desk reference for ColdFusion developers from novice to expert. 

CODE LISTING

The complete code listing for this article can be located at www.ColdFusionDevelopersJournal.com

About the Author

Jim Esten is the lead developer at WebDynamic, a company that specializes in CGI programming, installation and maintenance. He also heads ColdFusion and Perl development for StarQuest Internet Services, a southern California Web-hosting and dial-up services provider. He can be reached via e-mail at jesten@wdynamic.com.

jesten@wdynamic.com



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

<allaire> Alliance
Partner

CFDJ

www.sys-con.com/webinar

FORUM

bbboard/webboard.dll

Allaire Files for Initial Public Offering
(Cambridge, MA) – Allaire Corporation has filed a registration statement on Form S-1 with the Securities and Exchange Commission relating to the initial public offering of its common stock.

The company expects the offering to be completed in the first quarter of 1999, and will use the proceeds from the offering for general corporate purposes, including working capital. The offering will be made only by prospectus. **CFDJ**

The Igneous Group Introduces ITeam

(Santa Cruz, CA) – Until now, when given the task of implementing a team directory, Web managers have had to choose between a simple name-and-number listing or an expensive, custom-developed solution. ITeam directory software provides the best of both worlds. ITeam



replaces a one-dimensional team listing with a full multimedia directory that includes member photos, links to e-mail and Web sites, and more. Unlike a printed directory, a Web-based directory can be updated immediately, doesn't require printing and distribution and is available 24 hours a

Casio Kicks Off Their Online Shopping Site with AbleCommerce
(Battle Ground, WA) – Casio Inc., has launched their official electronic commerce site with Able Solutions' award-winning AbleCommerce.

Casio Online is a completely revised Internet site for Casio, the U.S. subsidiary of Casio Computer Company. With the introduction of Casio Online



you can purchase many of your favorite Casio products and accessories at www.casio.com.

Casio, a long-standing leader in the electronics industry, has a strong following of customers who will now be better served with the addition of the AbleCommerce site. AbleCommerce offers a powerful and flexible solution for the demanding and changing requirements of Casio Online.

For more information about Able Solutions visit their Web site at www.ablesolutions.com. **CFDJ**

day. It's also easy to install into an existing Web site, and can be customized to meet unique needs.

ITeam delivers colorful and comprehensive member profile pages. Each profile includes a photo, job information and e-mail addresses. It allows an unlimited number of phone numbers and mailing addresses. Several custom fields allow users to keep track of information that is important to a specific organization or group. For example, teams can keep track of their office phones, home phones, cell phones, faxes and pagers. Hyperlinks to members' e-mail addresses and Web sites allow users to simply click and send a message or view a site. Organizations can easily remove fields they don't want to show – employee

photos, for example.

A preview version of ITeam is now available at <http://igteam.igneous.com> and the ITeam Pro version begins shipping in January 1999, with a special introductory price of \$695 per server. ITeam Pro works with Allaire ColdFusion Application Server 3.1 or later, Microsoft Windows NT Server 4.0 (with Service Pack 3) and any ColdFusion-enabled Web server. For more information call 1 877 469-ROCK or visit their Web site. **CFDJ**

Catouzer Delivers Web Application Frameworks for Corporate Intranets

(Vancouver, BC) – Catouzer has released Catouzer Synergy 1.0, its Web application framework software. It provides employees with a tool for

instant, global, secure access to critical business information and collaboration. The mobile workforce can use Synergy for their information management needs without having to worry about time, computer systems or location.

Synergy is a Web application framework for instantly deployable corporate intranets. It's bundled with ColdFusion Application Server, the main component of Allaire's cross-platform Web application development system. Synergy's open architecture is designed for implementing existing and developing new ColdFusion applications specifically tailored to the customer's needs.

For more information about Catouzer visit their Web site at www.catouzer.com. **CFDJ**

RSW Software Introduces e-Load

(Boston, MA) – RSW Software, a business unit of Teradyne, Inc., has introduced e-Load, the newest addition to its e-Test suite of tools specifically optimized for testing Web-based applications. e-Load provides an easy-to-use and highly accurate



means to test the scalability of e-business applications prior to deployment.

e-Load's ease-of-use advantages over traditional load-testing tools make stress testing a more accessible and seamless part of the development process, allowing development and QA groups to

test early and often. Its automatic

response validation capabilities coupled with real-world usage scenarios make e-Load the most realistic load emulation tool available.

e-Load has been optimized for ColdFusion and is available for Windows-based platforms immediately. The starting price is \$19,950 for 100 virtual users.

For more information regarding RSW visit their Web site, www.rswsoftware.com, or call 508 435-8000. **CFDJ**


Live Software Announces <CF_Anywhere>

(Cupertino, CA) - Live Software, Inc., has announced a new product for developing Web applications. <CF_Anywhere> leverages the server-side Java technology in Live Software's JRun product to bring Allaire Corporation's ColdFusion Markup Language (CFML) to all platforms - including Linux, MacOS, HP-UX, AIX, Novell Netware 5, Alpha NT, IRIX and others - in addition to the standard platforms supported by Allaire.

By supporting a wide spectrum of platforms, <CF_Anywhere> assures the Web developer of complete cross-platform and cross-server compatibility, allowing development on one server/platform and deployment on another. <CF_Any-




where> gives ColdFusion users the ability to leverage all of the power and flexibility of Java using the familiar ColdFusion Markup Language.

<CF_Anywhere> will ship during the first quarter of 1999. Product pricing and an evaluation version is available at www.cfanywhere.com. For more information about Live Software, visit www.livesoftware.com, or e-mail info@livesoftware.com. 

Allaire Announces the Web Distributed Data Exchange

(New Orleans, LA) - The Web Distributed Data Exchange (WDDX), a new XML-based technology introduced by Allaire Corporation at the CNET Builder.com Live! conference, makes it easy for companies to exchange structured data between Web applications. WDDX gives companies the power to create "Web Syndicate Networks," where the content and commerce assets of any Web site can be exposed as services to other Web sites. This technology will enable new types of business on the Web, and increase interoperability between varying Web application platforms.

The WDDX SDK provides modules for JavaScript 1.x, ColdFusion 4.0, COM, Perl and Java. With the COM module, WDDX can be used from within Active Server Pages, Visual Basic, Delphi, Java, PowerBuilder, C++ and Microsoft Office.

Allaire can be reached for more information at 617 761-2000, by e-mail at info@allaire.com or at www.allaire.com. 




Infoboard Signs Agreement With Cyber Source

(Lynn, MA) - Infoboard, a Web-hosting company that specializes in providing developers with tools to build dynamic and commerce-enabled Web sites, has signed an agreement with Cyber Source. Cyber Source will now offer the Infoboard transaction processing services to its customers.

Infoboard provides ColdFusion 4.0 with its Web space. It also already offers credit card transaction software from Cybercash, which can be used with a variety of development tools, including ColdFusion. The Cyber Source product has a

Allaire Offers Computer-Based Training for ColdFusion


(Cambridge, MA) - Allaire Corporation has announced the availability of "SkillBuilding with ColdFusion," a 16-hour, self-paced, interactive CD-ROM that uses the latest multimedia technologies to provide developers with the core competencies of ColdFusion programming. During each unit, Allaire instructors present conceptual overviews, demonstrate a series of tasks using screen cams, animations, and walk-throughs, and coach the student to perform tasks independently. Quizzes and labs complement the unit instruction. At the end of the course, the developer will have demonstrated competency with ColdFusion by building an interactive Web application.

The course is available from the Allaire online store at www.allaire.com for an introductory price of \$295. 

Virtualscape Offers ColdFusion 4.0 Hosting

(New York, NY) - Virtualscape, a leader in advanced database Web hosting featuring ColdFusion and Active Server Pages, has announced the availability of Allaire's ColdFusion 4.0 hosting services to all new customers and to current customers who wish to upgrade.

ColdFusion 4.0 hosting is included with Virtualscape's NT Developer account, priced at \$49.95 per month. There is no additional cost for current NT Developer customers to upgrade to ColdFusion 4.0 hosting.


For more information about Virtualscape (Vscape International, Inc.) e-mail sales@virtualscape.com, visit their Web site at www.virtualscape.com or call 212 460-8406. 

Intermedia.NET Supports CF 4.0

(Palo Alto, CA) - Intermedia.NET, a Web hosting provider, has announced the support of Allaire's ColdFusion 4.0 to all customers. Support will be included in all hosting plans, which start at \$49 per month.


Intermedia.NET offers instant ODBC data source name registration and instant ColdFusion custom tag registration. Customers and developers hosting with Intermedia.NET will also benefit from the ability to manage verity collections for better search engine management.

Offering a full range of high performance Web-hosting solutions on advanced Windows NT and Internet Information Server 4.0 architecture, Intermedia was the first company to introduce Microsoft Transaction Server (MTS) support. Customers can work with tools such as Visual Basic, Visual C++, Java and InterDev to create MTS components for Web applications. Components can automatically be registered with the operating system using Intermedia.NET's advanced Site Management facility.

For more information about Intermedia.NET services write to webspace@intermedia.net or visit www.intermedia.net. 

simple interface that allows for separate address verification, credit card charging and sales tax computation.

In addition to ColdFusion, Infoboard supports various Netscape, Apache and IIS servers on UNIX and NT platforms. It also supports scripting and programming languages in addition to ColdFusion, as well as a number of "best-of-breed" applications for functions such as messaging and e-mail lists. Infoboard also offers SQL databases from Oracle, Informix, MS-SQL server and others, and is involved in back-end development as well.

For more information visit their Web site at www.infoboard.com. 



Dynamic Variable Length Forms

Taming multiple variables into submission

by Geoff Caras

Using forms is second nature for most ColdFusion developers. It's the method used to communicate information from the Web to back-end systems and databases. Forms are frequently created from a fixed set of information, as is dictated by functional requirements and business rules.

The use of forms is straightforward. In developing our product IGTeam, we found that we had several instances where the generic forms weren't ideal. We needed dynamic forms that would allow users to view and edit a variable number of entries in one step.

This article presents two methods for handling a variable number of elements in a ColdFusion form. Using dynamic forms allows ColdFusion developers to create forms that are populated from a database.

Most ColdFusion sites use forms for Web visitors to enter information into their databases. Many times developers use a second ColdFusion file to handle the actual database work. ColdFusion makes this process very easy and there are numerous examples of forms that submit to a second ColdFusion form in order to perform database functions and implement other business logic. Many ColdFusion developers also submit to the same file and use conditional logic to determine what action to take. For clarity, I've created two examples of this.

The Problem

IGTeam is a team directory application that stores information about each team member in the database. One of our design goals was to provide every member profile with an unlimited number of telephone numbers. Each telephone number has a type, a country code, an area code and a number.

At first, we created a simple form for editing phone numbers. This form had a submit, reset and delete button for each entry.

Although this solution was technically

adequate, we found that it simply didn't work well for most users. People who had many phone numbers (like Webmasters) found editing and clicking "submit" for each number tedious and annoying. Imagine that your area code has just changed and you now have a main number, a direct number and cell phone, beeper and home numbers to edit. You'd have to change one number, submit, return to the edit page, change the next number, submit and so on. This process is no longer user-friendly!

Example 1 (Listings 1, 2 and 3) shows the ColdFusion code for this approach. Figure 1, Example1.cfm, shows the drop-down menu used to select the team member whose phone numbers the user wants to edit. Once the member is selected, the form `_Example1Edit.cfm` (Figure 2) is active. This form lets the user make changes to a phone number. The final form, `_Example1Modify.cfm` (Figure 3), performs the update or deletion of the single selected entry.

Figure 1: Screenshot of Example1.cfm

Area Code	Telephone Number	Submit	Delete
831	469-7625	Submit	Delete
831	460-3979	Submit	Delete
877	469-7625	Submit	Delete
831	460-3971	Submit	Delete
831	460-3972	Submit	Delete

Figure 2: Screenshot of _Example1Edit.cfm

A Brief Note on Style

Over the years that we've developed ColdFusion forms, we've adopted a simple standard that makes it easy to determine the purpose of a file. Files that begin with an underscore (`_`) are forms that another form or file submits to. In Example 1 the file

Figure 3: Screenshot of _Example1Modify.cfm

that determines the member to edit is named Example1.cfm. The other two files start with an underscore because they require some information to be submitted to them in order to work properly. This convention is also enforced in the example files; if you type `_Example1Edit.cfm` directly into your browser, the form checks for the proper variables. If they're not present, the form uses `CFLOCATE` to redirect the browser to Example1.cfm. This process ensures that critical parameters are present when the form is run.

So What Did Users Want?

In testing our application, I found myself wanting to edit multiple numbers and have a single "apply" button submit all my changes at once. Our testers had tried to use the interface in this way and were confused when only one phone number updated at a time. It didn't take long to figure out that the simplest (i.e., easiest to implement) method, with "submit" buttons on each line, wasn't going to work.

We knew what not to do, so now the task was to discover a method that would allow for a variable number of fields in the form and a single "submit" button to make all the changes.

Example 2 shows simplified versions of ColdFusion forms from the IGTeam product that implements this approach. I've modified these to handle just area codes and telephone numbers and to make the examples smaller. Like Example 1, Example 2 has three listings (Listings 4, 5 and 6) associated with it. Example2.cfm (Figure 4) pro-

vides a drop-down menu to select the team member whose phone numbers the user wants to edit. Once selected, the form `_Example2Edit.cfm` (Figure 5) is active. This form lets the user make changes to all phone numbers. The final form, `_Example2Modify.cfm` (Figure 6), performs the update or deletion of ALL of the entries in the form.

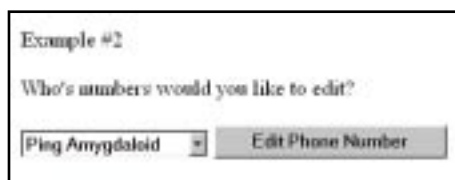


Figure 4: Screenshot of `Example2.cfm`

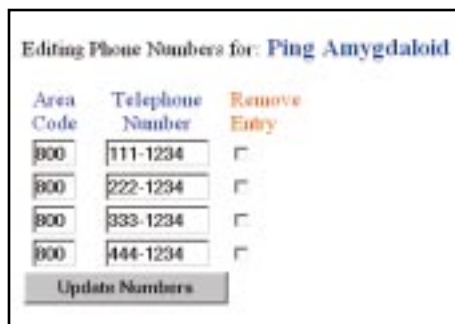


Figure 5: Screenshot of `_Example2Edit.cfm`

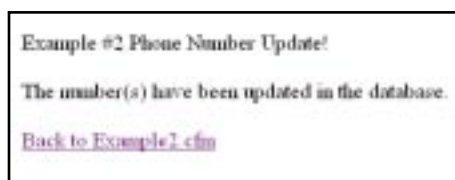


Figure 6: Screenshot of `_Example2Modify.cfm`

How Did We Do It?

To accomplish our objective we started with the following functional requirements:

1. The initial selection form must be populated by the database. This allows us to ensure that the user exists in the database and provides a more informative user interface.
2. A variable number of phone numbers must be viewed all at once. This information is populated from a database table.
3. Editing or deleting one or more phone numbers must be a single step.

Luckily, the engineers at Allaire knew we were coming. With a little digging we found just what we needed to solve the problem.

Because the data in the form is populated from the database, we needed a way to uniquely identify each form element. In this case, each telephone number received a unique ID. Whenever a user enters a new telephone number, it's assigned a unique ID. We designed our example database (Microsoft Access) with a primary key in each table that's always unique. Our design ensures that we can uniquely identify any row in the number table using this key.

Our solution has two parts that work together. The first part queries the database and populates the form information. We use `CFLOOP` because we want to iterate and change our counters as we process the query. The code for this is in `_Example2Edit.cfm`. We use our own counter, `EntryNum`, to determine the actual number of entries we've processed. Each time we enter the loop, `EntryNum` is incremented by one. Within the loop we must name each form element uniquely. We use `EntryNum` to number each element. For example, the name of the area code field is `ModifiedAreaCode_#EntryNum#`. Once the ColdFusion server has processed the form, this name becomes `ModifiedAreaCode_1` through `ModifiedAreaCode_x`, where `x` is the number of rows of data we found in the database. We also use this method to create a key variable for each entry. The key name corresponds with the entry number, and the value is the unique primary key for the telephone number. We can use this key to select an individual record.

Okay – The Names Are Unique

After we've done all this work, we also need to pass along a critical piece of information – how many there are! After the loop is finished, we use a hidden field, `EntryCount`, to store the number. `EntryCount` is assigned the value of our loop counter after the last entry is processed.

Now the Hard Part

Once the user has made modifications to the individual numbers (using `_Example2Edit.cfm`), clicking the "Update Numbers" button submits all this new information to `_Example2Modify.cfm`.

Building Variables

The code in `_Example2Modify.cfm` first checks to make sure that there's a variable called `EntryCount` because the form needs that information to work.

Then the form loops, starting at 1 and stopping at `EntryCount`. The loop uses an advanced feature of ColdFusion – the `Evaluate()` function. This function takes its arguments, left to right, returning (in this case) a string that has our variable name.

We use the `Evaluate()` function to get the value of the dynamically built variable. To create the name, we use the same base portion as with `_Example2Edit.cfm`: `ModifiedPhone_`. We add the value of the loop variable, `i`, at the end using pound signs: `ModifiedPhone_#i#`. This process gives us one of the variable names that we've passed on to the form, but we need the contents of the variable, not the name. The full expression for retrieving the value of the dynamic variable is `"#Evaluate("ModifiedPhone_#i#")#"`.

If the variable `i` is 3, the string returned by `Evaluate()` is `ModifiedPhone_3` – the exact name we need to determine a value. The outermost pound signs treat this expression like any other ColdFusion variable. `#ModifiedPhone_3#` gives us exactly what we need – the value contained in the variable. We assign the value to a set of temporary variables that are used for each pass through the loop.

Another standard we use is to make `q_name` indicate the current value of name. We use the `q_` in a variable to mean "current." (This is my shorthand to make naming easier.) For example, `q_phone` contains the current value of this entry's phone information. This is how we use temporary variables to modify or delete a given row in a database.

This method allows the final form to handle an arbitrary number of form elements and provides a flexible and dynamic – yet simple – method of naming the form elements while making sure each one is unique.

Is It Really Better?

This approach is more complicated, causes more database activity and updates the entire set of information – needed or not. Evaluating if it works in your environment



Figure 7: Screenshot of `IGTeam`

depends on the number of fields per record, the number of records your data is likely to incur and the usage pattern of your system.

I've included a screenshot (Figure 7) of how we use this interface in `IGTeam`. It makes the user interface work in a more natural way, and that's all the justification we need!

BTW

About the Author

Geoff Caras is a technical management consultant. He's also cofounder and president of *The Igneous Group, Inc.*, a consulting firm focused on helping clients apply strategic business perspectives to product development, marketing and management. Geoff can be reached at GeoffC@igneous.com.





File Upload Utility with ColdFusion

ColdFusion has a wealth of tags that are not related to database access

by Matt Newberry

ColdFusion is, arguably, the preeminent database middleware for NT-based Web sites. Somewhere around version 3.1 Allaire began calling their product an *Application Server*, implying it was more than middleware. In fact, ColdFusion 3.1 has a wealth of tags that are not related to database access. In addition, it provides a function library rich enough to remind veteran C programmers of the Standard Library – in design and intention at least, if not quite in breadth and depth.

I'll touch on several types of tags in the example application, but Table 1 summarizes the most powerful of them, which I call the *utility tags*. The function library includes string-handling routines, date/time functions, math functions, HTML and URL formatting, Boolean functions and much more. In short, the power to write Web-hosted versions of useful utilities is built in. I want to illustrate the use of some of the tags with a sample file upload application.

The Need for This Application

A digital imaging service bureau was receiving hundreds of megabytes of imaging files daily from their clients. They found that they needed to offer three or four different file transfer services so clients could pick the method that worked for them. "How," they asked me, "can we build a simple user-friendly file upload facility that everyone can use – Macintosh and Windows users, Photoshop fanatics and FTP experts alike?" They especially wanted people to stop attaching 35 MB TIFF files to e-mail, choking the mail server. It would also be nice to be notified about the upload so that rush jobs could be pounced on rather than languish in the upload directory for minutes at a time.

The browser seemed like the obvious starting point. Netscape Navigator's FTP client already allowed file uploads, and although it has a friendly drag-and-drop interface, it tended to baffle most first-timers. Besides, not everyone was using Navigator. A familiar applicationlike interface was needed, with form fields and an Upload button, a form that would work with any modern browser. The service bureau was already running ColdFusion for database access and they knew we could do a lot with the function library. We decided to take a look at those mysterious <CF> utility tags.

The HTML Form

The result was a file upload application based in ColdFusion. It consisted of an HTML form on the front end and a ColdFusion template at the back. As a courtesy to users, JavaScript opens a small floating window displaying the HTML upload screen (see Figure 1). This leaves the main browser window available for users who want to browse other screens during long uploads. The main browser window also displays a message for users of incompatible browsers, including alternate upload instructions. (See default.htm in the program listings.)

At the heart of the form is the file type input field, here named FileContents, which presents a text field and a browse button to the user. This little-known input type is not specific to ColdFusion in any way; rather, it is a 3.2 addition to the HTML specification. The full path of the file to be uploaded must be entered into the text box by typing or browsing. When the form is submitted, instead of submitting the filename as text, the browser opens the file and streams its contents to the server as the value of the

FileContents field. For this to work, the form's ENCTYPE must be set to "multipart/form-data" within the <FORM> tag. Stripped to its barest essentials, the form looks like Listing 1. (The complete HTML code is in UploadForm.htm in the program listings.)

The server must have a receiving process that is prepared to deal with this filestream; ColdFusion provides such a process. It is invoked by the <CFFILE> tag, as I'll demonstrate. The remainder of the form consists of two text input fields and a pair of radio buttons. The text fields, 'ClientName' and 'ClientPhone', provide identifying information to accompany the uploaded file. The radio button pair, named 'NameConflict', will be used to instruct <CFFILE> on how to handle potential filename conflicts on the server.

The ColdFusion Application

Now let's dissect the server side of the application (see UploadFile.cfm in the program listings).

ColdFusion has the <CFSET> tag for creating and initializing variables in server space. We begin the application by defining several variables that will be used as constants later on. This is just good programming practice. Putting these constants at the top of the template makes it easy to change them later, easier than if we had embedded the text strings at various places deep in the file. C programmers will recognize this as similar to the #define usage of that language. Alternatively, these text values could easily be kept in the database to be looked up in a ColdFusion query. Either way, we assign the values to variables, with <CFSET>, for later use in the program.

```
<cfset incoming    = "e:\ftproot\incoming">
<cfset mailserver  =
"mail.tinyflashunit.com">
<cfset jobs_email  =
"jobs@tinyflashunit.com">
<cfset phonenum    = "612-555-1212">
```

Next we set up a global error handler with the <CFERROR> tag; here Basic programmers will recognize a construct similar to 'On Error GoTo' in that language.

```
<cferror type="request"
    template="catastrophic.cfm"
    mailto="#fail_email#"
>
```

In the event of a catastrophic error anywhere in the file, <CFERROR> aborts processing of the current page and transfers control to the named error-handling template, catastrophic.cfm in this case. All we're allowed to pass is an e-mail address to respond to, but within the error handler a variety of error status variables are exposed. These may be dereferenced as #Error.StatusVar#. Together with the provided e-mail address, the status variables can be used to construct an intelligent error message for the user. A minimalist example is included in the source code listings as catastrophic.cfm.

We're really just trapping unforeseen critical errors here, such as disk-full on the server, so we can return some kind of message to users rather than leave them hanging.

After the error handler is established, we come to the heart of the application.

```
<cffile action="upload"
    filefield="Form.FileContents"
    destination="#incoming#"
    nameconflict="#Form.NameConflict#"
>
```

<CFFILE> is an extremely flexible tag that, through its "action" attribute, will allow you to perform almost any file-handling chore on the Web server. Here we're using the Upload action to receive the filestream being sent by the browser; the other possible actions are Move, Copy, Delete, Read, Write, Rename and Append.

Notice that we're passing the name of the form field, Form.FileContents, as the "filefield" parameter and not dereferencing it with #...#. <CFFILE> wants to know the name of the form field containing the filestream and will deal with it from there. Dereferencing #Form.FileContents# yields only a useless temp filename on the server.

The destination parameter is the name of the disk directory on the server where we want the file to reside. We assign the value of the variable, incoming, which we have previously set to hold a directory name. The "nameconflict" parameter tells <CFFILE> how to handle an upload when the same filename already exists in the destination directory. The two possible values are Overwrite and MakeUnique, and we let the uploader state his or her preference with the form radio buttons.

On completion of the <CFFILE> Upload action, a host of status variables will be set; they may be dereferenced as #File.StatusVar#. We need to save several of them for



Figure 1

later use as the values may be erased by subsequent invocations of <CFFILE>.

```
<cfset UploadSuccess = #File.FileWasSaved#>
<cfset FileWasRenamed = #File.FileWasRenamed#>
<cfset FileWasOverwritten = #File.FileWasOverwritten#>
<cfset NewFileName = #File.ServerFile#>
```

Now there's a housekeeping chore to take care of, and we need to exercise another action of the versatile <CFFILE> tag, the Rename action. If the uploaded filename already exists on the server and the uploader has requested the MakeUnique option on the form, <CFFILE> will create a new, arbitrary filename for the file just uploaded. This is not quite what we want. To preclude confusion, we'd prefer that the new file arrive with the uploader's original filename intact. We'll put our status variables to work here. Using ColdFusion's RandRange() function, the original conflicting filename is renamed by appending an additional extension in the form of a random three-digit number. Then the newly uploaded file can be renamed to the original filename. Listing 2 shows the conditional code.

Following a successful upload we'll generate two responses. A simple HTML response is returned to the client browser, informing the uploader that the file has been received. Then an e-mail notification is generated to inform the receiving party that a file has been uploaded by a customer.

<CFDIRECTORY>	Manage directories on the server
<CFFILE>	Manipulate files on the server
<CFFTP>	Send and receive files via FTP
<CFHTTP>	Communicate with HTTP servers
<CFLDAP>	Access directory services
<CFMAIL>	Compose and send e-mail
<CFPOP>	Retrieve e-mail
<CFSCHEDULE>	Schedule CF templates to run at specified intervals

Table 1

This latter job is handled by another ColdFusion tag, <CFMAIL>, that allows us to compose and send a mail message – in this case, of our saved status variables and the ClientName and ClientPhone form information (see Listing 3).

An upload failure is handled in exactly the same dual-response manner. See the source code listings for the failure code.


Summary

This sample application only scratches the surface of the capabilities that Allaire has provided. Do you want to build a Web-hosted version of a favorite network utility? Chances are that you can, with ColdFusion.

We could add a lot of automation to this

"Do you want to build a Web-hosted version of a favorite network utility? Chances are that you can, with ColdFusion."

simple file upload utility. We could write session results to an activity file, or log them to the database. Given login rights, <CFFTP> can forward uploaded files to any FTP server in the world. Using <CFSCHEDULE> we could automate an hourly, daily or weekly cleanup of the upload directory. Uploaded text files can be read, parsed, word-counted and displayed using the built-in string library.

If you're already using ColdFusion for database access, reserve your CGI programming in C or Perl for the particularly tough nuts; in many cases a little rapid development with ColdFusion will do the job. 

CODE LISTING

The complete code listing for this article can be located at www.ColdFusionDevelopersJournal.com

About the Author

Matt Newberry, an Internet database consultant in Minneapolis and St. Paul, specializes in Web site database integration with ColdFusion, Active Server Pages and Java Server Pages. He can be reached at matt@tinyflashunit.com.

matt@tinyflashunit.com

AbleCommerce Developer 2.6

Don't set up (e-commerce) shop without it

by Tom Taulli

True, it's possible to do just about anything with ColdFusion. But if there's already a solution on the market that's well tested, why reinvent the wheel?

Take AbleCommerce Developer 2.6. With AbleCommerce you don't have to spend thousands of hours of programming time to create a sophisticated e-commerce store. In a sense, AbleCommerce gives you the best of both worlds. If you want to customize the store, you can use ColdFusion and not have to learn another scripting language. If you don't want to do much programming, you can use over 300 wizards and templates to create a quick commerce site. You don't have to know ColdFusion at all – or even HTML.

As with any good e-commerce tool, you can build your storefront from your browser. Your store can be virtually any size – from thousands of products to just one. There are two basic parts to AbleCommerce: System Administration and Merchant Administration (see Figure 1).

System Administration

Store Groups: In each Store Group you can have one or more stores (depending on how many your license allows). By clicking on a Store Group, you can view a list of the stores and add or edit products (see Figures 2 and 3). You can also make global changes to your settings (see Figure 4). You can change things such as shipping classes (flat fee per order, flat fee plus weight rate, etc.), payment methods, such as Visa, MasterCard,



AMEX, Discover, etc. and tax rates (see Figure 5).

AbleCommerce supports major payment processing systems, e.g., CyberCash, ICVerify and Authorize.Net (see Figure 6), and it even supports Secure Electronic Transactions (SET), the most secure form of e-commerce.

You can also change the log-file options. For example, you might want the server to generate daily, weekly or monthly log files that you can then analyze using WebTrends or other third-party software products. (AbleCommerce is bundled with Web-

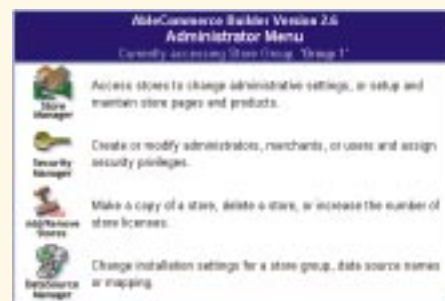


Figure 1: The main administration menu

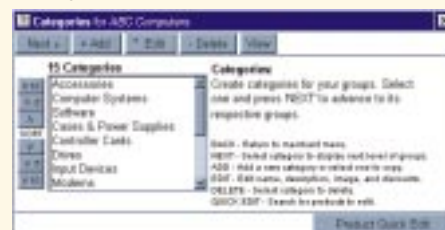


Figure 2: A display of categories

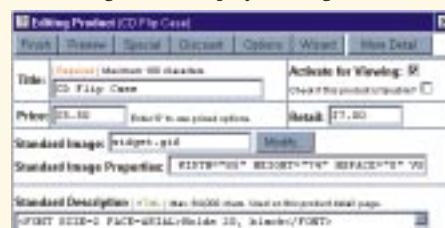


Figure 3: A product editing page

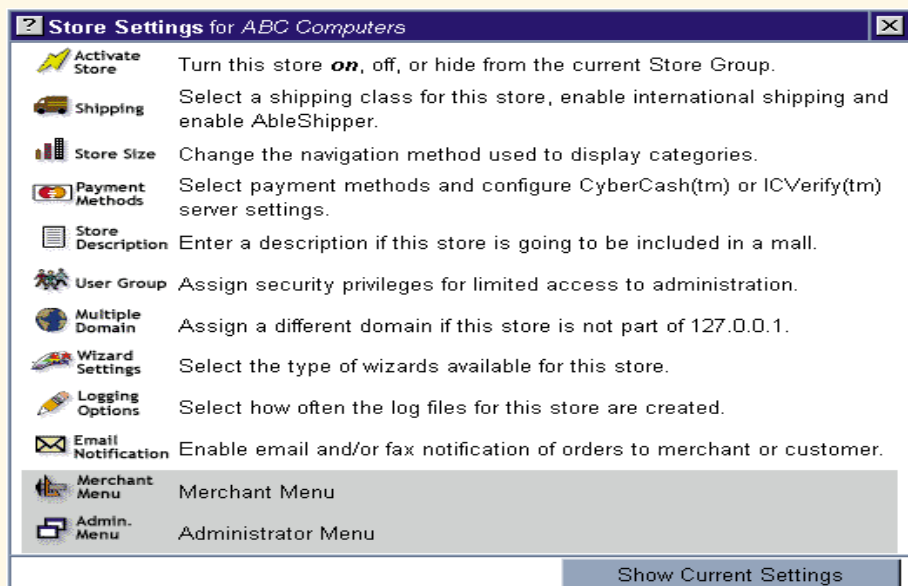


Figure 4: The global settings menu

Trends, which is a top-notch product for traffic analysis.)

ProtoFax Gateway (another bundled product with AbleCommerce) allows you to securely fax order confirmations to merchants. In other words, a merchant can get orders without going through the Internet.

Security Manager: This is where the user information is stored. The first category of users has no administrative access. These users either access a page of the site and are tracked by IP address, browser type and a random cookie ID, or complete a checkout that requests a user name, random password and contact information.

The second user category has administrative access. These users are either merchants who have access to the Merchant Administration page or systems administrators who have full access.

Data Source Manager: This allows you to add, edit or remove Store Groups. Each Store Group has its own content database as well as a user security database. The stores in each group can then have their own domain names and security certificates.

There's support for MS Access 7 and SQL Server 6.5, as well as for any ODBC-compliant database.

Merchant Administration

Styles: This gives you free rein to change the look and feel of your store in terms of fonts, colors and navigational images. This makes life easier, especially if your store has many products.

Web Pages: As the name implies, you can customize each Web page to your specifications, either by writing the HTML (manually or by using an editor such as FrontPage) or by using wizards.

Inventory: Inventory is based on a hierarchical structure of three levels – Category, Group and Product – all of which are flexible. For example, you can move a Group and its products to a different Category or move products to a different Group.

As is typical for most stores, each product can have a graphic. This helps to motivate the buyer.

AbleCommerce has also included Font F/X Text Renderer, which helps you create professional-looking images.

Discounts: You can set discounts, based

Add new Tax Rate for ABC Computers

Select a state: Alabama (AL)

Tax rate: 0.00000 Required
Percentage in decimal format.
(EXAMPLE: 0.065 = 6.5%)

Are shipping charges taxable in this state?
☐ Yes
☒ No

Update

Figure 5: Add a tax rate page

Payment Methods for ABC Computers

Accepted payment methods

☒ Visa ☐ CyberCash 2.x
☒ MasterCard ☐ ICVerify
☒ American Express ☐ CyberCash 3.x
☒ Discover Card ☐ Authorize.Net
☒ Fax Order In
☒ Call Me
☐ Purchase Order

☐ No Real-Time Processor

Allow Merchant to delete Orders?
☐ Check to Allow

Inventory Management - Recommended for real-time payment processing

☒ None: This store has no Inventory Management.
☐ Sell Only In Stock: This store will only sell products that are currently in stock.

Update Next >

Figure 6: A selection of payment methods

on a percentage or dollar amount off a product, to categories, groups and/or products. You can also set minimum and maximum purchase requirements.

Orders: This is a list of unprocessed orders you can process and then either delete or archive.

Reports: The four main reports are sales by date range (shows gross sales), recent shoppers, browser types (shows a list of browser types and the percentage of use) and log files (a list of available log files).

Comments: This will allow the user to send you comments about your service – a valuable information source for any merchant who wants to improve service.

Conclusion

AbleCommerce Solutions was founded

in February 1995 and has a well-tested product that works well with ColdFusion. Sites such as Casio, Hasbro (the company Intranet) and even Jimmy Buffet's Margaritaville (the official Buffet Store) use the software.

Although the product is great, there are some shortcomings. I'd like to see a general-purpose Store Wizard that creates a simple store, and it would be nice to have several step-by-step demos in the documentation for creating a store.

All in all, AbleCommerce will help you create your online store very easily, and without sacrificing functionality. **CDM**

About the Author

Tom Taulli is the CEO of Blueprint Interactive, a company that develops Internet applications for the enterprise. You can reach him at ttaulli@bpia.com

ttaulli@bpia.com



Taming the Cookie Monster

***Hysteria about cookies is wildly distorted.
Here's how to handle them in
your applications***

by Bob Siegel

Setting a cookie on your visitor's computer makes it easier for you to have a fully functional Web site. Using a cookie, you can keep track of your users' visits, their shopping cart and anything you care to record. With server-side programming (read "ColdFusion") you can customize pages and create order forms – and anything else you care to program. Because there is still resistance to cookies on the part of some Web users, the methods I'll describe are calculated to keep the cookie out of the user's face as much as possible.

Setting the Cookie

Setting the cookie couldn't be simpler with CF. The one shown below will last for one week.

```
<CFCOOKIE name="Kookie" value="#UserID#"
expires=7>
```

The UserID can be almost anything, but I much prefer a randomly generated string of 10 alphanumeric characters. The important thing is that no user will ever have the same ID as another user. Thirty-six (a–z plus 0–9) alphanumeric characters to the tenth power will yield 3,656,158,440,062,980 different IDs (that's 3.6 quadrillion), so we really don't need to worry about duplicates. Generating the ID in your CF code will save a trip to the database to get the value of an auto-increment or counter field. Using only alphanumerics will ensure that the ID will behave properly for all Web and database uses.

It's best to set the cookie only when it's actually needed. Doing it on your home page can create a bad first impression. If you are to maintain a shopping cart, it would be best to set the cookie only when visitors are putting something in it. If they

have cookie warnings set, it'll be clear to them exactly why you're setting it.

Getting the Cookie

Determining later that your visitor has your cookie, and its value, is equally simple. Testing for your cookie will NOT cause a cookie warning to pop up.

```
<CFIF IsDefined("COOKIE.Kookie")>
<CFSET currentUserID = COOKIE.Kookie>
</CFIF>
```

While some CF-ers prefer to use ColdFusion client variables (which depend on behind-the-scene cookies for their implementation), I avoid them like the plague. When we started with client variables at my commercial Web site, the rate of ordering products dropped to near zero. Admittedly, my clientele might be more sensitive to privacy issues, but setting a single client variable (if the browser is set to show cookie warnings) will pop up not one but TWO cookie warnings, each with the ominous notice that the cookie will last for FORTY YEARS. Any change in the value of a client variable will likewise pop up warnings. Also, earlier CF versions use the NT registry to store the client variables, which made my Webmaster very unhappy.

Using the Cookie

When you use the cookie properly, there's never any reason to set more than one per user or per session, depending on how you set the expiration. This is important if you want to keep the cookie as unobtrusive as possible. The thing to remember is that the value of the cookie is only a key for you to look up whatever you have stored in your database about this user. The cookie should not be used to store any-

What Are Cookies and Why Do We Need Them?

A cookie is usually a very small piece of data written to your hard disk by your browser at the request of some Web site you visited. Its main function is to allow the domain that put it there (and no others) to recognize you the next time you come back. The important thing to remember is that the next time may be only seconds after the first time.

Because of the way the Web functions, all that ever normally happens on the Web is that first you ask to see a page, then the server at the Web site you're visiting responds by sending you the page. That's the end of it. Unless something extra is done, the server can't tell that you're the same visitor who requested the same or some other page even a few seconds earlier.

If you are to have any meaningful interaction with your visitors (like maintaining a shopping cart), you need a cookie. There are other methods of doing the same thing, but cookies are the easiest and most reliable way of doing it. That's why they were invented.

thing other than the ID. Who knows? If your visitor happens to look at his cookie file and finds some bit of data about him that he can decipher, he might not be very happy about it. Exactly what you want to store is entirely a function of what you want to do on your site, and only you will know what you have stored. Yes, you can set dozens of cookies on your visitor's machine, each with a piece of information, but this is obtrusive and unnecessary, and if your visitors are at all cookie-sensitive, it's sure to send them fleeing.

A word of caution is in order here. Some programmers can't resist the temptation to put up text such as: "You have visited this site 12 times, and the last time you visited was on 7/4/98, and you bought 4 cases of Viagra and a cooling fan." DON'T DO IT! Such comments are sure to give your visitors a VERY creepy feeling. ("What else do they know?")

Getting Rid of the Cookie

You might think you should expire the cookie, as follows, once you don't need it:

```
<CFCOOKIE name="Kooki e" value=""  
expires=Now>
```

This is not a good idea, as it can also cause a cookie warning to pop up. Just let it expire naturally.

Who Stole the Cookie?

You may find that your cookie hasn't been set (IsDefined) at a certain point in your page sequence when under normal conditions it should have been (you think). This is usually because your visitor has refused the cookie, but it could be that he or she is using an ancient browser that doesn't support cookies. A third possibility is that your visitor has bookmarked one of your pages and the cookie he or she once had has expired. I can't suggest what you should do in all cases for all Web sites, but you should be aware of these possibilities and try to program for them. Unfortunately, it probably won't be possible to determine why the cookie hasn't been set.

Common Cookie Bugs

You need to know that the browser uses information in the HTTP headers (which you and your users normally never see) to



set the cookie. Knowing this will help avoid two common bugs.

Bug 1: You cannot test for the cookie being set on the same page as the page containing the CFCOOKIE. Whatever your IsDefined test returns will have nothing to do with whether the cookie has been set successfully. This is because nothing will happen cookie-wise until your visitor has received the page. You have to test for it on a later page.

Bug 2: You KNOW you set the cookie, but it's vanished. The culprit here is the use of CFLOCATION on your page after the CFCOOKIE. Relocating the user to a different page effectively blows away the headers on your current page. Thus the browser never sees them – and never sets the cookie. This is not a CF problem but a Web problem. This can be avoided by using a JavaScript Location, but I haven't tried it.

The Cookie Monster

It seems that the media, no doubt encouraged by the marketers of antiookie software, have created the hysteria about cookies. There have been reports such as:

"When you register at a Web site, you cough up a lot more than your username and password. Cookies suck information from your hard drive." – *Wired Magazine*, 2/98.

At www.davecentral.com/6069.html we find: "Cookie Terminator will clean up hidden cookies and prevent Web sites from snooping around your machine."

And from *PC Magazine*, found at

www.zdnet.com/products/content/pcmg/1706/-284070.html: "...cookies may track your activities on the Web in ways that violate your privacy."

Such "reporting" is frequently seen in print and, of course, on the Web.

Now that we understand cookies, we can see that these are wildly distorted, if not just plain wrong. The one from *PC Magazine* is almost true, but cookies (or, more correctly, the server-side software that uses the cookie to recognize you) can track your activity only on a single Web site or between other Web sites that have made special arrangements among themselves. You can't be tracked on the entire Web, as the phrase "your activities on the Web" ominously implies.

The truth is, by using a cookie (or one of the alternative methods) you can record only information that the visitor has freely given to you, either by entering something into an input box or by recording the visitor's progress through your Web site. They definitely cannot "suck information from your hard drive" or "snoop around your machine."

By the way, I just went to *Wired.com*, and no less than FOUR cookie warnings popped up just while their home page was loading.

Alternatives to Cookies

I said earlier that the value of the cookie is only a key for you to look up whatever you have stored in your database about this user. The cookie is the mechanism by which you can recognize a visitor when he or she requests his or her next page. Two other methods can be used to maintain a key that persists from page to page.

The URL Appendage

Method: An ID, exactly like the ID a cookie can save, is generated, but it's passed from page to page in the form ``. All links on all your pages must be dynamically replaced by your CF code to have the correct ID for your user. The ID can also be passed in a hidden form field if a page happens to be a form. (Note that "ID" is just what I chose to call it, and can be the identifier of your choice. You refer to the "ID" in your CF as URL.ID.)

Limitations: Every page on your site must be dynamically generated. It can't be a static HTML page or it won't have the correct ID. This could cause a load problem on your server, but most likely won't. I've heard knowledgeable estimates put the overhead of 20-40 milliseconds to pass a cfm page when no processing other than doctoring a few URLs is being done.

To be fair, it does have one advantage. Such a URL with an ID can be saved by your visitor in a bookmark and will work as long as you haven't cleaned the user out of your database.

IP Address

Method: The user's IP address (in CF it is known as "CGLREMOTE_ADDR") is used as the ID. It's an up-to-15 character string in the format "123.123.123.123".

Limitations:

1. Certain ISPs, most notably AOL, bizarrely and perversely change their users' IP addresses DURING their login session.
2. Many ISPs, while not changing addresses during a session, can't be depended on to supply the same one to each user every time he or she logs on. This is because they just take the next one available from a pool they've been assigned.

This is okay for all uses of an ID except those that have to recognize the visitor the next day, as opposed to only a few minutes later.

Again, to be fair, the IP address works most of the time, given the limitation in Item 2. It even works for AOL for a while; AOL told me they routinely change users' IP addresses only (!) every 10 minutes.

In-the-Field Experience

At my commercial Web site we use the IP address unless we can detect an AOL (and certain other ISPs) user, or unless the user checks a box that he or she wants the "shopping cart to last for a whole week" or is having a problem about "losing your shopping cart." Then we use a cookie.


So far this has been a manageable compromise. We hope that as time goes on and the hysteria dies down, we can be more upfront about using cookies, and use them for more visitors.

When we switched to ColdFusion from Visual Basic cgi programming, we started with client variables, which was a disaster, as I mentioned before. We then switched to regular cookies with an explanation of why we needed them and how they were harmless. Ordering improved, but was still off by about 60%. We tried

more and better explanations, hoping to soothe any cookie-phobia, but only made things worse. We then switched to our current IP/cookie method but things did NOT improve.

It was only by deleting the offending word from our site entirely that sales returned to normal. We still set them for some users, but notice that our wording about the cookie checkbox doesn't mention the dreaded word "cookie."

The Future Fortunes of Cookies

This situation will change eventually, like so many other things about the Web. But for now, we have to use cookies carefully to avoid scaring away the uninformed. And it wouldn't hurt to explain patiently to all who'll listen exactly what cookies can and cannot do. Just don't try to do it on a Web site where you want to set a cookie. 

About the Author

Bob Siegel is the founder and chairman of the NYPC ColdFusion Special Interest Group (cfsig.org) and has written articles for many publications. He's been computing professionally for almost 30 years and has been an independent software developer for the past six. His CF clients have included AT&T and Lucent Technologies. Bob can be contacted at bs@interport.net.

 bs@interport.net 

Fusion FX

www.fusionfx.com



Building Custom Tags Using ColdFusion

Developers can encapsulate functions, snippets of code and entire applications in a portable, modular format

by Ashley King

Have you built a custom tag yet? If you haven't, you're missing one of the most exciting features of ColdFusion 3.0. Custom tags are the ability to extend ColdFusion Markup Language (CFML) with custom tags. For the first time, developers could encapsulate functions, snippets of code and even entire applications in a portable, modular format. A quick browse through the over 400 custom tags in the Allaire Tag Gallery attests to their popularity and flexibility. But just when you thought it couldn't get any better, it did.

In ColdFusion 4.0, enhancements in custom tag functionality enable developers to build nested custom tags. These nested tags can be associated with each other so that data is passed easily between tags. While this is a significant advance in CFML, there's more functionality than meets the eye. We'll cover this functionality, and we'll build the <CF_CoolMenu> custom tag to demonstrate some of the possibilities. Our CoolMenu tag will build a JavaScript menu that will display a set of items and perform an associated action when an item is clicked. Let's compare building a tag the old way with the new custom tag model.

How the New Custom Tags Are Different

With the old custom tag model, passing sets of data to a tag is a cumbersome process. The CoolMenu tag illustrates a problem that many ColdFusion programmers have faced. For each menu item two pieces of information are passed: a link and the text for that link. These sets of data are passed to the tag using two lists, as shown in Listing 1. The data is interpreted properly only if the related elements in each list are in the same position in their respective lists. If we accidentally changed the order of the Name list by placing CFDJ in front of Allaire, the two links would be switched.

In the new custom tag model descendant tags are used to collect these sets of data by grouping them together, as shown in Listing 2. Descendant tags are simply "tags within tags" that are related in function. Descendant tags are ColdFusion templates just like any other custom tag. The outer tags, called the *base tag* or the *ancestor*, are also a single ColdFusion template. This template actually executes twice, once before the descendants are executed and once afterwards. Because of this new way of doing things, some architectural changes are made inside the custom tag to make it aware of where it is in the application page.

What They Look Like Inside

The new custom tag model features a new variable scope called ThisTag. Some special variables in this scope are provided for data exchange and flow control in custom tags. This scope is not read-only, so both reading and changing the values of these variables is allowed. The first variable you'll want to know about in this scope is called ThisTag.ExecutionMode. The value contained in this variable tells your custom tag what section of code should execute. Let's see this in action.

In our CoolMenu tag we need to accomplish three things: set up the JavaScript, gather the data sets we'll use in the menu and then build the menu itself. The descendant tag accomplishes the middle step, gathering the data sets. This means that our base tag needs to know if it's executing before or after the descendant tag. Listing 3 shows the basic detection structure we use to determine the part of the application page that needs to execute. When ThisTag.ExecutionMode equals "start," the base tag is executing before the descendant. Likewise, when ThisTag.ExecutionMode equals "end," the descendant has completed its execution.

For our two tags, CoolMenu and Cool

Menuitem, to work together as base tag and descendant, we must first associate them. By associating the tags we allow the values of the descendant tag's attributes to be available to its ancestor. We can then take the data entered as attributes in the CoolMenu-Item tag in Listing 2 and use it in the CoolMenu tag in its "end" ExecutionMode. We associate the two tags by adding a new tag, CFASSOCIATE, to the beginning of the descendant custom tag as shown in Listing 4. By doing this we're telling ColdFusion to send the attributes of the current tag to the specified ancestor, the CoolMenu tag. Once the ancestor tag takes over control during its end execution mode, the attributes from all of its associated descendants are available in the variable ThisTag.AssocAttrs. Let's simplify this a bit.

You've noticed that we've used the terms *base tag* and *ancestor* almost interchangeably so far. There is a difference between the two. An ancestor is any tag that contains other tags between its start and end tags. A base tag is an ancestor that has been explicitly associated with a descendant using CFASSOCIATE. Only the base tag has access to the attributes of its associated descendants. Here's how our example shows this in action.

Listing 2 shows all our tags working together. First the ancestor tag, CoolMenu, is executed. The code that runs is contained in the first CFCASE shown in Listing 3. Then the descendant tag, CoolMenuitem, is executed. The code that runs is the CFASSOCIATE tag shown in Listing 4. This runs twice, once for each time the CoolMenuitem tag is placed in the application page. Next the ancestor tag, CoolMenu, is executed once again, but this time the code that runs is the last CFCASE shown in Listing 3. In this example each tag runs twice! The application pages are reused automatically, which means less code for you to write.

Building the Base Tag

We've built our basic structure in Listing 3 for our base tag. To this listing we'll add a section that initializes the attribute variables we'll need later on in the application page. Any variables set in the "start" execution mode will remain active in the "end" mode. Next we'll add the JavaScript that's necessary to set up the menus. We do this with a

CFHTMLHEAD tag that inserts a SCRIPT tag in the HEAD section of the document. The actual script that builds the menu object is very large, so we'll just use SRC= to point to the menu.js file. Remember that this file must be placed in the same directory that the CoolMenu tag is called from. Everything we've added to the listing will execute before our CoolMenuItem tags are called.

Now we'll need to add the code that goes in the "end" execution mode. First we'll add the JavaScript to define the instance of the menu object we just set up. Next we'll use the variable ThisTag.AssocAttrs to get the data structures passed as attributes to the CoolMenuItem tag. This variable is a record set containing a new kind of variable called a *structure*, which is similar to associative arrays in Perl. Listing 5 shows how we can loop through this record set to retrieve the attributes for each descendant. There will be one record for each CoolMenuItem tag that executes. Finally, we construct the actual completed JavaScript and insert it into the HEAD section of the document using CFHTMLHEAD.

Building the Descendant

In our example the descendant is used only to retrieve attribute values and nothing else. Because of this, we'll need only a small amount of code. Descendants can be as com-

plex as you want, but we'll shift the complexity to the base tag. Listing 6 shows our CFASSOCIATE tag and our initialization section. One thing we could do here is move some of the JavaScript building to the descendant, or even some more complex validation of the attributes.


Because we've associated the CoolMenu and CoolMenuItem tags, any variables we set in the Attributes scope are automatically available to the base tag in the ThisTag.AssocAttrs variable. There are more low-level ways of exchanging data between these tags that allow direct access to the variables in the base tag, but for our example this simple method of exchange works well. With that, our CoolMenu tag is complete!

Converting Older Tags

If you've written any custom tags for ColdFusion 3.x, no conversion is necessary. In fact, you can still write tags using the old model with no changes at all. At times it may be desirable to allow your tags to detect whether or not the developer has added an end tag to them. In this manner you could build a tag that behaves differently if there is an end tag. The ThisTag.HasEndTag variable will tell you if an end tag exists. You could then check the execution mode and execute the appropriate piece of code. If there is no

end tag, the value of ThisTag.ExecutionMode will always be "start."

So why use this new custom tag model? It gives you the flexibility to break up complex tags into sets that are more intuitive to work with. Large sets of attributes can be entered using descendant tags, and you can even access the results of code executed between your start and end tags. In effect, you could design your own sublanguage! Our CoolMenu tag is just a small example of what can be done with the new custom tag model.

There are many more features in this new model that we haven't touched on, such as low-level data exchange, looping within your base tag (making your tag act like a CFLOOP), manipulating the generated content and more. All in all, the power of custom tags in ColdFusion 4.0 has been increased exponentially. This new power will allow you to build more concise and efficient application pages and will set your development pace to warp speed. 

About the Author

Ashley King currently works for Allaire. He has coauthored The ColdFusion Web Application Construction Kit (second and third editions) and Advanced ColdFusion. He can be reached at ashley@mrpost.com.

ashley@mrpost.com

Listing 1.

```
<cf_cool Menu
  name=i favoritesi
  names=i Allaire, CFDJi

actions=i location='http://www.allaire.com', location='http://www.sys-
con.com' i>
```

Listing 2.

```
<cf_cool Menu
  name=i favoritesi>

<cf_cool MenuItem
  name=i Allairei
  action=i location='http://www.allaire.com' i>

<cf_cool MenuItem
  name=i CFDJi
  action=i location='http://www.sys-con.com' i>

</cf_cool Menu>
```

Listing 3.

```
<cfswitch expression=i ThisTag.ExecutionMode#i>

<!ó- This gets run before the descendant tags ó->
<cfcase value=i starti>
  <!ó- Start section code ó->
</cfcase>

<!ó- This gets run after the descendants ó->
<cfcase value=i endi>
  <!ó- End section code ó->
</cfcase>
```

```
</cfswitch>
```

Listing 4.

```
<cfassociate basetag=i CF_COOLMENUi>
```

Listing 5.

```
<!ó- Build an empty array just in case there are no descendants ó->
>
<cfparam name=i thisTag.assocAttrs default=i #arrayNew(1) #i>

<!ó- Gather descendant data ó->
<cfloop index=i i from=i 1i to=i #arrayLen(thisTag.assocAttrs) #i>

  <!ó- Get the attributes for a menu item ó->
  <cfset thisItem = thisTag.assocAttrs[i]>

  <!ó- Add to JavaScript ó->
  <cfset javascript = javascript &
i #attributes.name#.addMenuItem(i i #thisItem.name#i i,
i i #thisItem.action#i);i>

</cfloop>
```

Listing 6.

```
<!ó- Associate attributes ó->
<cfassociate baseTag=i CF_COOLMENUi>

<!ó- Initialize attributes ó->
<cfparam name=i attributes.action default=i i>
<cfparam name=i attributes.URL default=i i>
```

CODE LISTING

The complete code listing for this article can be located at www.ColdFusionDevelopersJournal.com

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

<allaire> Alliance Partner



An ad in the ColdFusion Marketplace can bring you more business! Expose your product or service to over 30,000 registered ColdFusion users.

For more information, contact Robyn Forma at 914 735-0300 or robyn@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.

You can reach Able Solutions at www.ablecommerce.com or at www.ablesolutions.com.

11700 NE 95th Street, Suite 100, Vancouver, WA
360 253-4142

ALLAIRE CORPORATION

At Allaire, our focus is to empower developers with the tools and knowledge to deliver on the promise of the Web as a platform for crucial business applications. We offer a wide range of flexible programs, including professional education, consulting, technical support and partner programs that complement our existing documentation and online developer's center.

You can contact Allaire at www.allaire.com.

One Alewife Center, Cambridge, MA 02140
888 939-2545

CATOUZER INC.

With 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 apps, which can be fully integrated and managed under the Application Services Layer (ASL).

For more information contact Catouzer at www.catouzer.com.

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

EPRISE CORPORATION

If your customers are looking for a content management solution, Eprise Participant Server 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. Get in touch with us today (www.eprise.com) to learn more about the Eprise Participant Server FastStart Kit for Allaire ColdFusion Developers.

1671 Worcester Road, Framingham, MA 01701
800 274-2814

THE IGNEOUS GROUP, INC

The Igneous Group provides custom solutions for e-commerce business information systems and dynamic content publishing. We build systems to your specifications and integrate them with your existing backoffice data and business infrastructure. We can help you get your site online, or identify how to make it function more efficiently.

For more information, check out The Igneous Group at www.igneous.com.

541 Seabright Avenue, Santa Cruz, CA 95062
877 469-ROCK

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.

Contact Intermedia, Inc., at www.intermedia.net.

953 Industrial Avenue, Suite 121, Palo Alto, CA 94303
650 424-9935

LIVE SOFTWARE

The power of Java and the simplicity of ColdFusion, <CF_Anywhere> gives ColdFusion programmers the ability to leverage all the power and flexibility of Java using the familiar ColdFusion Markup Language (CFML). <CF_Anywhere> was perfected for CF developers, CF administrators, ISPs and everyone.

For more information about <CF_Anywhere>, go to www.cfanywhere.com.

5703 Oberlin Drive, Suite 208, San Diego, CA 92121
408 996-0300

VIRTUALSCAPE

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

Virtualscape can be reached at www.virtualscape.com.

215 Park Avenue South, Suite 1905, New York, NY 10003
212 460-8406

COLD FUSION MARKETPLACE

GET YOUR OWN!

Subscribe Today
and receive the
"CFDJ Digital Edition"

FREE

at

www.COLDFUSIONJOURNAL.com

two
years
\$89⁹⁹
12 issues

save
\$18

one
year
\$49⁹⁹ don't miss a
single issue!
6 issues

\$69 one year Canada/Mexico
\$79 one year all other countries



1 800-513-7111

or subscribe online for faster service
subscribe@sys-con.com

USER GROUP NEWS

User Group	e-mail	Contact Info
Florida User Group	Dale@reddesign.com	710 94th Avenue North, Suite 310, St. Petersburg, FL 33702
Hawaii CF Developer	huifeng@hawaii.edu	808 351-3904
ACFUG	cameronc@mcrae.com	McRae Communications 770 460-7277 ext. 232 – phone 770 406-0963 – fax
CFSIG	bs@interport.net	bs@interport.net
Bay Area ColdFusion Users Group	nathan@roundpeg.com	www.bacfug.org
Atlanta CFUG	tim@cmserver.com	430 Tenth Street NW, Suite N206, Atlanta, Georgia www.cmserver.com/atlantacfug/404 892-9300
CF_Employ_Me	ryan@halfzero.com	Ryan Phipps
CFDEV	stephenschuster@clarkmhc.com	606 288-1474
SCCFUG	lchalnick@prprpr.com	Leon Chalnick Professional Presence Providers, Inc. 562 491-0212
SoCalFusion	smcrae@performanceweb.com	www.PerformanceWeb.com/SoCalFusion.cfm
TeamCFM	mrowan@netsense.net	231 Old Tower Hill Road, Wakefield, RI 02892 877 TEAM-CFM
UTCUG	utcfug@enhtech.com	Enhanced Technologies, Inc. 175 North 200 West, Suite 09, Provo, Utah 84601-2844 801 373-4001
VACFUG	vacfug@enhtech.com	Enhanced Technologies, Inc. 6422 Grovedale Drive, Suite 301E, Alexandria, VA 22310-2534 703 924-0301
Tek-Tips	dtrocino@tecumsehgroup.com	www.tek-tips.com
fusoids	drew@gotrax.com	Trax Communications, Inc. 10220 NE First Place, Suite 200, Bellevue, WA 98004
CFUG	drew@projectalpha.com	312 NW 10th Avenue, Portland, OR
JCFUG	gus@catouzer.com	http://b-factory.com/JCFUG
Fusion	coldfusion@goblue.com	210 495-9321
California-Bay Area BACFUG	mike@digitalchef.com	707 967-0540 ext. 24
NCCFUG	robis@ania.com	Robi Sen
CFUG	steve@ateaze.com	@eaze Productions 37 Hibbert Street, Suite #2, Arlington, MA 02476 781 641-1661
CFUG	gary@creativeis.com	Gary Arndt Creative Internet Solutions 401 North 3rd Street, Suite 590, Minneapolis, MN 55401
NEW JERSEY NJCFUG	pomarlec@sharpsec.com	Cindy J. Pomarlen c/o NJCFUG 2467 Route 10 East, Bldg. 7-8B, Morris Plains, NJ 07950
NYCFUG	mdinowit@houseoffusion.com	Michael Dinowitz
North Carolina	m@secretagents.com	Steve Nelson
DFWCUG	Josh@dowdell.com	Josh Dowdell
DC-CFUG	sdrucker@figleaf.com	Steve Drucker, Figleaf Software Washington, DC 202 797-5477
AK	akieran@hq.nasa.gov	409 3rd Street SW, Suite 800, Washington DC 20024 202 651-8549 – phone 202 651-8510 – fax

Allaire 4

www.allaire.com

Intermedia

www.intermedia.net