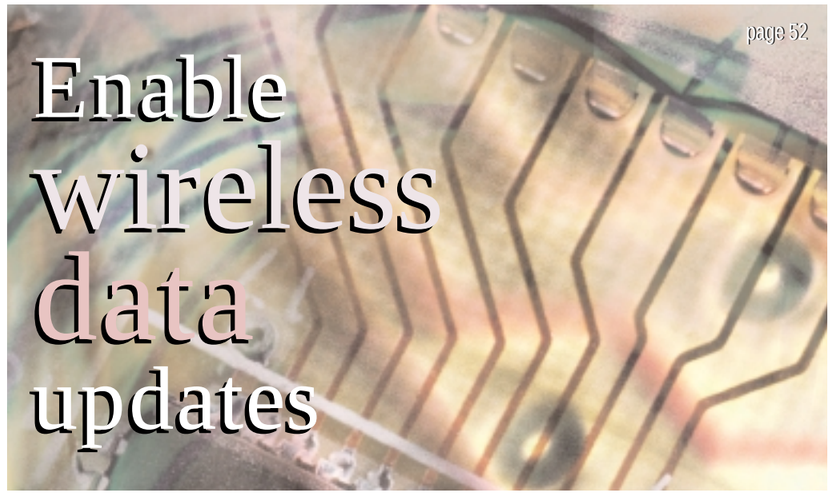
CREATING *UNWIRED* SITES

U.S. $8.99 (Canada $9.99)

ColdFusion Developer's Journal

ColdFusionJournal.com

February 2001 Volume: 3 Issue: 2



SYS-CON MEDIA

?

ColdFusion Developer's Journal

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

EDITOR-IN-CHIEF ROBERT DIAMOND
ART DIRECTOR JIM MORGAN
EXECUTIVE EDITOR M'LOU PINKHAM
EDITOR NANCY VALENTINE
MANAGING EDITOR CHERYL VAN SISE
ASSOCIATE EDITOR BETTY LETIZIA
ASSOCIATE EDITOR JAMIE MATUSOW
EDITORIAL INTERNS SUZANNE AUGELLO
PRODUCT REVIEW EDITOR TOM TAULLI
TIPS & TECHNIQUES EDITOR MATT NEWBERRY

WRITERS IN THIS ISSUE
CHARLES AREHART, ANDREW CRIPPS, ROBERT DIAMOND, RALPH FIOL, BEN FORTA, HAL HELMS, EBEN HEWITT, MARGARITA LAPLAZA DE ANDROUIN, GUY RISH, JOSEPH SCHMULLER, CHRISTIAN SCHNEIDER, BRUCE VAN HORN, JOHN WILKER

SUBSCRIPTIONS
SUBSCRIBE @SYS-CON.COM
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT.
SUBSCRIPTION HOTLINE 800 513-7111
COVER PRICE $8.99/ISSUE
DOMESTIC $79/YR. (12 ISSUES)
CANADA/MEXICO $99/YR
OVERSEAS $129/YR
BACK ISSUES $12 EACH

PRESIDENT AND CEO FUAT A. KIRCAALI
VP, PRODUCTION JIM MORGAN
SENIOR VP, SALES & MARKETING CARMEN GONZALEZ
VP, SALES & MARKETING MILES SILVERMAN
ADVERTISING ACCOUNT EXECUTIVE ELIZABETH LEHMANN
ADVERTISING ACCOUNT EXECUTIVE RONALD J. PERRETTI
ASSISTANT CONTROLLER JUDITH CALNAN
CREDIT & COLLECTIONS CYNTHIA OBIDZINSKI
ACCOUNTS PAYABLE JOAN LAROSE
ADVERTISING ACCOUNT MANAGER ROBYN FORMA
ADVERTISING ACCOUNT MANAGER MEGAN RING
ASSOCIATE SALES MANAGER CARRIE GEBERT
ASSOCIATE SALES MANAGER CHRISTINE RUSSELL
SALES ASSISTANT ALISA CATALANO
VICE PRESIDENT, CIRCULATION AGNES VANEK
CIRCULATION MANAGER CHERIE JOHNSON
ART DIRECTOR ALEX BOTERO
ASSOCIATE ART DIRECTOR DINA ROMANO
ASSISTANT ART DIRECTOR CATHRYN BURAK
ASSISTANT ART DIRECTOR LOUIS F. CUFFARI
GRAPHIC DESIGNER ABRAHAM ADDO
GRAPHIC DESIGN INTERN AARATHI VENKATARAMAN
WEB DESIGNER STEPHEN KILMURRAY
WEB DESIGNER GINA ALAYYAN
WEB DESIGN INTERN PURVA DAVE
SYS-CON EVENTS MANAGER ANTHONY D. SPITZER

EDITORIAL OFFICES
SYS-CON MEDIA, INC. 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 **FAX:** 201 782-9600
COLDFUSION DEVELOPER'S JOURNAL (ISSN #1523-9101) IS PUBLISHED MONTHLY (12 TIMES A YEAR) FOR $79 BY SYS-CON PUBLICATIONS, INC., 135 CHESTNUT RIDGE RD.,MONTVALE, NJ 07645

POSTMASTER
SEND ADDRESS CHANGES TO:
COLDFUSION DEVELOPER'S JOURNAL
SYS-CON MEDIA, INC.
135 CHESTNUT RIDGE RD.,MONTVALE, NJ 07645

WORLDWIDE DISTRIBUTION
BY **CURTIS CIRCULATION COMPANY** 730 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

SYS-CON MEDIA

Can You Escape the Wireless Revolution?

BY **ROBERT DIAMOND**

This editorial has been written in a way unlike any article you've ever read in this magazine. As a testament to the focus of this issue – "ColdFusion and Wireless" – I'm typing it (albeit painstakingly so) on my RIM Blackberry 957. Three sentences in, I can tell you that it's by no means the easiest way to type out 700 or so words, and it's not something I'd recommend (*later note*: neither would my thumbs!) to other writers. But the bottom line is that it's doable, and although it might not be the best way, it's there and the technology behind it is being developed and clamored for like nothing before.

I recently participated in an online chat with some of the Internet industry's leading consultants, and the topic turned to wireless technologies – certainly not a surprise since it's the latest buzzword. The discussion centered around whether clients are really asking for wireless additions to consulting projects or if it has yet to hit prime time in terms of demand.

The general consensus was that it's still more or less an "afterthought" in large consulting projects, where the company asks for it late in the game as an "à la carte" add-on rather than something that's considered and planned out as part of the main project. However, the trend is moving in a direction that includes wireless access as part of the initial discussions and project development.

Everyone in the online discussion seemed to agree that although the companies are asking for it, they aren't exactly sure what it is they're asking for. Everyone knows the wireless buzzwords, and "remote access" sounds like a great feature. When they try to figure out what to access, and how to wireless-enable it – that's when the confusion kicks in.

Consultants and clients end up in befuddling conversations, trying to iron out what's viable technologically to access over a cell phone or Palm Pilot, and then trying to figure out if it would actually be of any use. Often when the consultants come up with something they could easily implement, the client finds it completely useless. On the flip side, every time the clients come up with something they'd love to be able to do, the consultants lose sleep as they attempt to figure out how in the world they could possibly deliver it.

The reason is that the technology is still as fuzzy as most people's understanding of what it can (or perhaps should) do. In this issue we attempt to clear up that confusion – at least from a developer's standpoint. From a user's standpoint, it'll still be confusing. But here at ***CFDJ*** we do what we can.

We've got some great articles in this month's issue that demonstrate the "whys" and of course the "hows" of creating unwired sites. First up is a piece entitled "M-Commerce: A Practical Approach to Usability" by Ralph Fiol, a case study about mobile.actibetier.com. The article covers the architecture of the mobile application as well as some design guidelines for WAP and WML projects. We've also got a great piece on "Updating Databases Wirelessly," which covers CF, Fusebox, and WML, written by Margarita Laplaza de Androuin and Joseph Schmuller. Of course that's just the tip of the iceberg. You can also dip into the usual great roundup of material from ***CFDJ*** regulars Ben Forta, Hal Helms, Charles Arehart. and others. Read on…

ABOUT THE AUTHOR
Robert Diamond is editor-in-chief of **ColdFusion Developer's Journal** *as well as* **SYS-CON**'s *newest magazine,* **Wireless Business & Technology**. *Named one of the "Top thirty magazine industry executives under the age of 30" in Folio magazine's November 2000 issue, Robert is currently a senior in the School of Information Studies at Syracuse University.*

ROBERT@SYS-CON.COM

Allaire Technology Roadmap

BY ANDREW CRIPPS

This article describes Allaire's technology roadmap. Allaire started off with a single product, the ColdFusion Web application server. Now they have many products, and there's an increasingly strong relationship between them. Allaire is using a range of code names to refer to its products, and it can be difficult to remember which products are which and which code names refer to which releases.

In this article you'll learn about the products and their code names, how the products relate, and the features Allaire is promising in new versions. With this information you'll be armed to answer questions such as:

- *Should I stick to programming in ColdFusion Markup Language (CFML) or switch to Java?*
- *Should I build some functionality myself or wait for the next product release from Allaire?*

At the Allaire Developer Conference (November 5–8, 2000) Jeremy Allaire reiterated his belief that CFML remains the "fastest, easiest, and certainly the most fun way to build Web applications." He also said that ColdFusion remains at the center of Allaire's product strategy. How is Allaire planning to keep ColdFusion at the center while at the same time promoting their Java application server, JRun?

A Web Platform

Allaire is trying to create what they call a *Web platform* – a foundation on which you can build Web systems in the same way a computer operating system is the foundation on which you build software applications. The pillars that create this foundation were articulated in the "Spectra Technology Overview" white paper as:

- Web systems management
- Packaged systems or applications
- Visual tools
- Application servers

But at the conference and in the March 2000 Allaire "Technology Roadmap" white paper, this set of pillars had been reformulated as:

- Application frameworks (changed from packaged applications)
- Visual tools
- Application servers

Between papers Allaire did some fresh thinking. The Web systems management pillar is still important to Allaire's strategy, but it's not presented as a separate pillar, and the packaged applications pillar has changed to application frameworks because the product offered for this pillar, Spectra, was being viewed by customers as an out-of-the-box solution. The old term *packaged applications* gave people the impression that they could take Spectra and use it as is. In fact, Spectra provides an object-oriented framework with which to build applications, so the pillar was renamed *application frameworks*.

Allaire Objectives

The technical direction for Allaire's products comes from their business and user objectives, articulated at the conference by Jeremy Allaire. Let's explore them.

Ease of use: Allaire products should have simple interfaces and be simple to use. Jeremy Allaire gave a demonstration of what he meant by this by creating a simple ColdFusion application (using technology from Voxeo) that responded to telephone calls and provided a telephone interface to sound files.

Productivity: Allaire creates tools that increase developer productivity. They offer strong development environments, powerful tags and programming language tags, and extensions that help developers produce software solutions quickly.

Open: In the last year customer interest has shifted to companies offering products that adopt or adhere to open industry standards and architectures. Allaire is embracing open standards and recommendations such as Java and XML.

Empowering: One design goal in Allaire's application servers and frameworks is to package complex technology so it's easy to use and deploy. Allaire makes it easy for developers to get their work done, even when it involves complexity. Operations such as transferring data between languages such as JavaScript and CFML, or calling Java Servlets on a remote machine, are made easy through custom tags or ColdFusion tags.

Accessibility: Allaire's business goal is to make products that are inexpensive and broadly adopted and deployed. Some companies try to be profitable by charging a small customer base a lot of money for their services and products; Allaire has the opposite view – make the products inexpensive and effective, and they'll be broadly adopted.

Product Roadmap

In this section we look at the products that make up the technology pillars of Allaire's Web platform. Figure 1 shows the product roadmap for application servers (ColdFusion and JRun), application frameworks, and visual tools. The long-term trend is to provide a single application server – code-named Pharaoh – that runs both CFML and Java applications.

The switch to a single-application server from the current split offerings of ColdFusion and JRun will occur after ColdFusion 5.0 in a release called "Neo" (ColdFusion 6.0), demonstrated at the conference. With Neo your CFML code will run on a Java engine, but you'll still be able to run your ColdFusion applications without change on the same server (see sidebar, "How Can CFML Run on a Java Engine?"). A simple application to calculate prime numbers was run on the ColdFusion engine and on the Java engine (Neo). Neo proved to be about three times faster than the ColdFusion engine in this case.

The move to Java is shown in Figure 2. At present, only the ColdFusion and Spectra products rely on the ColdFusion engine. Both products will be moving to a Java-based engine in the next couple of releases. Technologies from JRun, ColdFusion, Harvest, and Tron will all contribute to the Pharaoh product release.

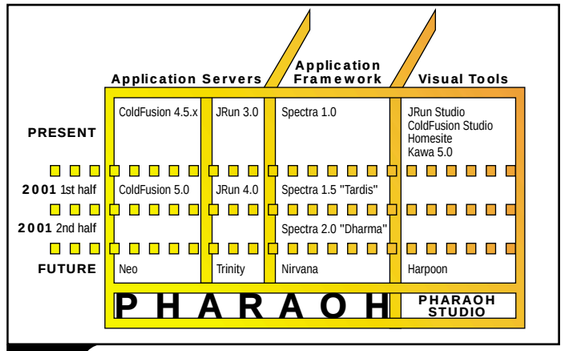


FIGURE 1 Product roadmap

Allaire Products

So far we've looked at the technology pillars of Allaire's Web platform. In this section we look ahead to what's planned for the products that form those pillars.

Web Application Servers

Allaire started business offering only ColdFusion. Developers could write Web applications in CFML. But Allaire acquired JRun from LiveSoftware in 1999, adding an important component to their Web platform. Before the acquisition, with only ColdFusion as their product offering, Allaire met considerable opposition from some potential customers because it had only a "proprietary" Web application server and language. With JRun, Allaire had a strong contender for people looking for an open standards Web application server. JRun 4.0 will comply with the Java 2 Enterprise Edition (J2EE) 1.3 specification and the Enterprise JavaBean (EJB) 2.0 specification from Sun Microsystems.

With JRun and ColdFusion, Allaire is reaching two broad sets of Web developers: Java developers and ColdFusion developers.

The challenge for Allaire, which they've successfully addressed in their technology roadmap, is to integrate these two apparently separate worlds of Java and ColdFusion without losing developers in either camp.

ColdFusion 5.0 and Neo: Allaire has planned the next two releases of ColdFusion: ColdFusion 5.0 and ColdFusion 6.0 (Neo). In the first half of 2001 Allaire will introduce ColdFusion 5.0, which is built on the existing ColdFusion code base. After ColdFusion 5.0, the Neo release is planned. Neo will be based on the JRun engine and will run CFML and Java applications.

In ColdFusion 5.0 you can expect to see a host of new features, some borrowed from the Harvest initiative, some a result of feedback from the Allaire developer community. The following features were demonstrated at the conference:

- Server management features (from the Harvest initiative)
- Context-sensitive help
- User-defined functions and function libraries
- Ability to query against existing query sets
- Partial-page output
- Server-side graphics engine

The new server management features include archiving ColdFusion applications as .car files, restoring ColdFusion applications, monitoring server and application health, analysis of ColdFusion log files, and a new way to protect your ColdFusion source code through encryption.

Context-sensitive help will be added to ColdFusion 5.0's administration section. If you want to know how to perform a certain task or get definitions of various settings, you'll no longer need to dig for the manual – you'll click on the help icon.

User-defined functions are a major addition to ColdFusion 5.0. You'll be able to create libraries of your own (and others')

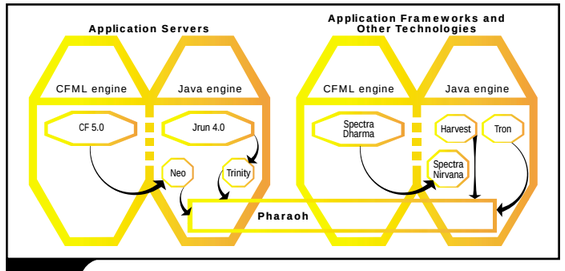


FIGURE 2 The road to Java-based products

functions and include them whenever you need to do so. For example, you could define a function called #myprefix(string,">")# that returns a string prefixed with ">". This has been one of the most widely requested additions to ColdFusion, and it's nice to know that Allaire listens to its customers.

Another much-requested feature is the ability to query existing query sets, meaning that you can execute a CFQUERY statement and then execute another CFQUERY against the results of the first. One application of this would be to query a database for a product set when the application first starts up, and then use a second CFQUERY to query the memory-resident product set obtained from the first query. Another application would be to obtain results from one database with the first CFQUERY, and then combine them with results from a second database in the second CFQUERY.

Partial-page output makes it possible to stream Web pages to a browser as the pages are being generated by ColdFusion.

The last ColdFusion 5.0 feature demonstrated at the conference was the server-side graphics engine. Although Allaire didn't reveal whose technology they were using, it looked as though ColdFusion 5.0 will include software to generate Macromedia Flash graphs. One company offering this technology right now is CORDA Technologies in their PopChart product.

JRun and Trinity: JRun 4.0 is to be released in the first half of 2001. Trinity will follow. JRun offers many features, including adherence to open standards, the ability to run on a broad range of platforms, a distributed object architecture, and support for unicode. JRun 4.0 also provides support for JSP custom tags that hide Java programming complexity for the developer and a WAP (Wireless Application Protocol) gateway that allows developers to quickly create and test WAP and Wireless Markup Language (WML) applications.

Pharaoh – the future application server: Pharaoh is the code name for the initiative that wraps a number of future technologies under development at Allaire. At its core Pharaoh will be an application server with a Java engine that runs both CFML applications and applications built with Java. Pharaoh will also include Harvest and Tron (and possibly Spectra) technologies. With Pharaoh, you can combine CFML, JSP, servlets, and EJBs. You'll have a framework for creating multitiered applications.

Pharaoh will be backward compatible with both ColdFusion 4.x and JRun 3.x applications. Yet it will be fully compatible with Sun's J2EE specification and therefore appeal to developers wanting to create applications with Java and use technology that takes advantage of open industry standards. An example architecture of a multitiered Web application running on Pharaoh is shown in Figure 3.

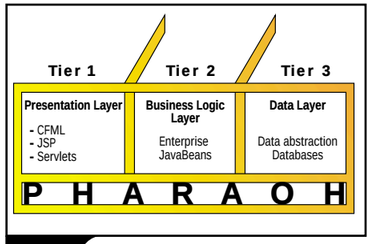


FIGURE 3: Multitiered architecture

Visual Tools: Kawa, Harpoon, Pharaoh Studio

The visual tools pillar covers all of Allaire's development tools. There were two announcements of interest at the conference:

1. Allaire has acquired Kawa from TekTools, Inc.
2. Allaire is working closely with Macromedia on Harpoon – software that lets you create Macromedia Flash components from ColdFusion pages.

Kawa: Kawa is an integrated development environment for Java from TekTools. Allaire is shipping Kawa version 5.0 as an Allaire product. An important component of Kawa for Allaire is the ability to create and edit EJBs. Kawa also provides class browsing, syntax highlighting, project management, context-sensitive Java Development Kit (JDK) help, and a powerful debugger. It's likely that the underlying capabilities of Kawa will be harnessed to the familiar JRun Studio user interface.

Harpoon: The Harpoon initiative is a collaborative effort between Macromedia and Allaire. The goal is to give ColdFusion developers the ability to create Flash components from ColdFusion. Macromedia's Flash version 5.0 is likely to be increasingly popular on the Web as a technology for creating user interfaces. Using ColdFusion tags, you specify data sources for data that's passed to Flash. The Flash elements are dynamically generated with your data and can be manipulated by dragging and dropping.

Pharaoh Studio: In the future, Allaire will combine their HomeSite (HTML editor), ColdFusion Studio, JRun Studio, and Kawa technologies into a single IDE (integrated development environment) code named Pharaoh Studio. Pharaoh Studio will provide a seamless environment for Java, CFML, JSP, JavaScript, XML, WML, and so on as well as integrated tools for wireless application development and creating server-side Java applications. In addition, Pharaoh Studio will provide greater support for building Spectra applications.

Application Frameworks: Spectra, Dharma, and Nirvana

Spectra has been incredibly successful. Launched in 1999, it has accounted for a significant percentage of Allaire's revenue. Spectra was initially pitched as a packaged application for content management, e-commerce, and personalization. This positioning has caused some confusion in the marketplace, however, as customers imagine they can deploy an e-commerce or business Web site out of the box.

At the conference, Allaire clearly repositioned Spectra as an application framework, which is much closer to the truth. With Spectra you get a set of integrated services that allow you to rapidly create Web applications. Spectra is targeted to developers and provides the framework for rapidly building Web systems that need content management, personalization, and e-commerce capabilities.

Spectra 1.0 was extremely ambitious and included a wide range of innovative ideas. Allaire has continued to work hard on Spectra and plans to release Tardis (Spectra version 1.5) in the first quarter of 2001. Later in the year, Dharma (Spectra 2.0) is planned. No dates have been released for product launch. Dharma will still use the ColdFusion engine. After Dharma, the Nirvana release is planned, and Nirvana will use an underlying Java engine (see Figure 4). Spectra technology will also contribute to the Pharaoh application server effort.

Web Systems Management: Harvest

Allaire has made the lives of developers simpler with ColdFusion by achieving their objectives of packaging complexity and ease of use. What Allaire has done for Web systems developers, they'd like to do for Web systems administrators. "Harvest" is the code name given to the technology initiative to create an easy-to-use Web systems management tool. Allaire acquired the seed technology from Bright Tiger in 1999.

Harvest is aimed at meeting the growing demand for highly reliable Web systems that operate 24/7. Allaire identified the need for a set of management utilities that would take the pain out of managing these systems. Harvest will build on

Allaire's current Web systems management technology, such as clustering (for both ColdFusion and JRun), Cisco LocalDirector integration, administration of distributed components in JRun, Web browser–based administration tools, remote development services, and scriptable deployment and replication.

For example, if you have a cluster of servers providing highly reliable service for your Web applications, a cluster of database servers, and some technology in place to provide load balancing and failover, Harvest will give you system monitoring and configuration capabilities from a single console.

Harvest will ship as part of ColdFusion 5.0. Dave Gruber, senior product manager for ColdFusion and Harvest, demonstrated the following features at the conference:

- Application and ColdFusion server monitoring
- Archive and restore capabilities for backup and fast deployment
- Log file analyzer
- New encryption scheme for CFML

You can create probes that check on various aspects of the health of the ColdFusion server or application. For example, you can create a probe that checks if a page is missing from an application. If the probe detects a problem, the administrator will see a warning.

The second major feature is archive and restore. You can take a snapshot of an entire application (including server and application settings and databases, if you need them) and archive the entire ColdFusion application in a .car (ColdFusion archive file). The archive files can be deployed to a ColdFusion server; in Dave Gruber's example he repaired a broken application by restoring the last good version of it.

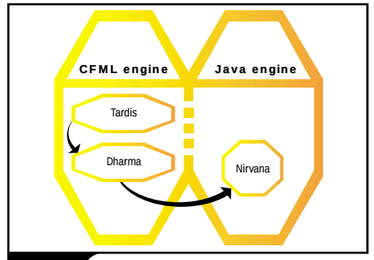


FIGURE 4: Spectra's move to Java

The ColdFusion server creates several log files, and with the log file analysis tools that ship with ColdFusion 5.0 you'll be able to combine data from several log files and search for different error conditions or keywords across files in a given time frame. You can also obtain statistics on the number of errors of a given type over a given time range.

The management tools for a single server are extended in Harvest to allow you to manage applications that run on a distributed environment. For example, you can deploy your ColdFusion archive not just to a single server, but to an entire server farm. Harvest technology not included in ColdFusion 5.0 will debut in Pharaoh.

Tron

Tron is the code name Allaire has given to their technology that addresses complexity for B2B applications. Tron embraces XML as a language of data interchange between two Web systems. It provides a framework for building workflow applications that need to exchange data and services. It addresses the need businesses have to exchange data and participate in multiparty processes.

One of the problems with exchanging data between applications is that the data may be represented in different XML Schemas. Tron provides a data mapping facility so that fields from one XML Schema can be easily mapped to those in another. Tron gives the developer a means to create "nodes" that perform simple pieces of processing, as well as "interpreters" that can transform data.

In addition to data interchange businesses need to be able to share and use distributed services. The World Wide Consortium has a working group for XML Protocol, which is the successor to SOAP (Simple Object Access Protocol). Tron uses XML Protocol as the vehicle to provide distributed application functionality over HTTP. On your system, as a developer, you create APIs to functionality that you want to make available to a remote system. Tron simplifies the creation and syndication of your services.

Summary

Allaire has a well-defined technology roadmap that takes them well beyond 2001. The company has carefully selected and acquired products and technologies that strengthen their product offerings.

The move to Java and the corresponding industry perception of commitment to open standards that this move entails is of central importance to Allaire's strategy. With a popular and powerful Java engine (JRun) that can be sold independently – and sold as the underlying engine for ColdFusion and Spectra – Allaire has aligned itself with the open standards movement and is squared off against Microsoft in the marketplace for Web platforms. Microsoft has largely chosen to pursue the Web platform market with its own set of tools and technologies.

The move to Java doesn't mean that ColdFusion developers need to learn to program in Java or JSP, however. With future application server offerings, ColdFusion developers will be able to continue to create and deploy ColdFusion applications and see performance and reliability benefits. But at the same time, those developers can choose to create and deploy Java applications.

The combination of an open standards, Java-based application server, a strong visual integrated development environment, and an application framework providing services for rapid application development puts Allaire in a strong position in the Web platform market. In addition, Allaire is spending research dollars on technology that's soon to be in high demand by businesses, such as Tron and Spectra's ability to create B2B solutions quickly and Harvest's ability to administer and manage large Web systems.

It's going to be an interesting road to follow with Allaire, with many improvements that will simplify the creation of Web applications. Keep the technology roadmap handy!

About the Author

Andrew Cripps is manager and technical lead at evolutionB (www.evolutionb.com), currently on a project using Internet and Web technologies. He has over 12 years of experience in the computing industry, with masters' degrees in both computing science and philosophy.

andrew@evolutionb.com

HOW CAN CFML RUN ON A JAVA ENGINE?

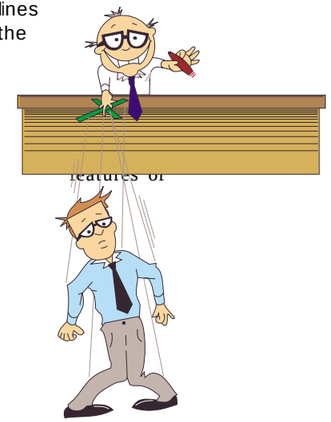
Sun Microsystems has created a specification for JavaServer Pages (JSP), a simple way to use the Java language along with HTML without having to precompile. JSP is extensible. You can create JSP custom tags to abstract complexity from your JSP page. JSP custom tags work in a similar way to ColdFusion custom tags. Allaire's approach to getting CFML to work on a Java engine is to try to write the expressions and tags of CFML as JSP custom tags. When the CFML page is requested on Neo, your CFML code is actually seen by Neo as a JSP page with Java code in it. JSP pages are compiled (on the first request) into Java bytecode and executed as Java Servlets on the Java engine.

Best Practices in ColdFusion Development

For neophytes and veterans

BY EBEN HEWITT

In recent years we've witnessed a trend in which organizations gather, codify, evaluate, and disseminate seemingly endless lists of "Best Practices" – guidelines that help promote the success of their business, the consistency of their actions and, presumably, the general cheerfulness of their employees.



Best Practices are cherished by neophytes and mentors alike for many reasons. For beginners (or those toiling fiendishly into the night, alone in their basements, who can't help but cackle, "It's alive!" every time their application runs without crashing), Best Practices offer a coherent and consistent mode of communication with fellow coders who might inherit their work. They also offer guidelines that hasten the debugging process, and they create a compendium of development tips and tricks that can be time consuming to find on your own. For more seasoned developers, Best Practices offer a refreshing prompt to review their habits – as Hal Helms suggested in "Making Assertions" (***CFDJ***, Vol. 2, issue 10) – and provide an excellent excuse for engaging in witty and urbane debate with one's peers.

These particular Best Practices are divided into five categories: General, Security, Portability/Modularity, Speed/Performance, and Teamwork/Readability. In the spirit of pursuing a provocative American trend, I offer this list of Best Practices for ColdFusion coding and development, which is the strange fruit of many conversations with many developers. My hope is to continue that conversation, and I invite your responses as well as your own Best Practices.

General Practices

When debugging an application, set a simple variable and use <cfoutput> to display it on your processing page.

For instance, you might write <cfset Status = "Okay"> in the offending template, then write <cfoutput>#Status#</cfoutput> in the page in question. If you see the word "Okay" when you view your page in a browser, ColdFusion is processing your application at least up to that point. Move the <cfoutput> tags around within your application, and you can pinpoint exactly how far your application is processing, depending on whether or not your output is displayed.

This is also a good way to ensure compatibility of your applications across browsers. For instance, Internet Explorer 5.5 handles table display differently than IE 5.0 does. I've seen well-written applications that simply don't display in IE 5.5. This tip was the light at the end of the tunnel.

You can use a related technique within <cftransaction> blocks. For instance, set the status variable as above, and write a conditional clause around your <cftransaction> <cfquery> block. Set the status to "Not Okay" within a <cfcatch/cftry> block where the <cftransaction> is "Rollback".

```
<cfif status IS "Okay">
<cftransaction action="commit">
<cfquery…></cfquery></cfif></cftrans-
action>
```

Your output will indicate the status of your database transaction.

Use the poor man's Spectra.

While this little idea certainly doesn't cover the many excellent features of Spectra, doing the following will help keep your clients – or your non-IT staff – happy when frequent content updates are an issue. Use a custom tag for formatting and general page display, and <cfinclude> all your content. For instance, when working on a page called "Aboutus.cfm", you might create a custom tag called <cf_layout> or <cf_table>, which allows easy and consistent specification of its attri-butes. You can title the headers of your tables, specify the colors and borders, and size them up. Then write a file called "AboutusContent.cfm", which is only plain text and no code. In a cell within your table call <cfinclude template = "Aboutus Content.cfm">. You can specify fonts with an external stylesheet or from within your custom tag, and you've got instant, consistent formatting and total separation of code from content. When it's time for others to update the content, they don't touch a line of code. Note that there's a slight performance penalty for using <cfinclude> as well as for any custom tag.

```
<cf_LogoTable>
 <tr><cfoutput>
  <td align="right"><font face=#font#"
size=#medsize#">
Username:</font></td>
  </td><input type="Text" name="User-
name" size="#bigsize#"></td></cfoutput>
 </tr>
</cf_Logotable>
```

where the file LogoTable.cfm defines the fonts and sizes as cfparameters and also begins the table by defining a <tr> that contains the company logo, a background color, and more. It's a way of making a hybrid stylesheet/HTML that's easy to reuse.

You can also easily borrow Spectra's Universally Unique Identifier for Content Objects with the ColdFusion "CreateUUID" function as follows: <cfoutput>#CreateUUID()#</cfoutput>. It's also native to Transact-SQL as "NewID()". They both generate a 128-bit value, (almost) guaranteed to be a unique value for any computer on a network because the last six digits are generated from the node number for the IEEE 802 identifier on your machine's NIC. Kewl! (Notice that there are a couple dozen CF functions like this one that have an exact counterpart in SQL. Use the CF functions when possible, however, for reasons that follow.)

Always scope your variables.

Perhaps it's a little thing, but I've found that referring to #Form.My Variable# rather than #MyVariable# makes it easier for other developers to follow your work, and it cuts down on debugging time. As a side note, remember that when working with <cfmodule>, the code of the module or custom tag is executed in its own process, outside the scope of the calling template. (Thus the module has no access to variables in the calling template.)

Security

Don't use hidden fields to pass any sensitive or important variable (e.g., "price" or a limitation on record set returns).

While it's less of a problem with ColdFusion, it takes seconds to hack a page written in Perl or any CGI/server-side language that passes hidden form field variables. (Hacking 101: simply save the page source as an htm file, change the hidden variable to a price or limitation you like better, and pass your new local page to the absolute URL of the processing page.)

> "The sine qua non of many ColdFusion applications is a database; therefore, slow database = slow CFServer, but not the other way around

In guest book or other form, don't allow *<CF* as user input.

User input forms allow what my wife refers to as an "attractive nuisance"; hooligans have an opportunity to run code (cfdirectory, cffile) from your text fields. Make sure your application knows to return a tsk-tsk if it spots such input. Check for this by placing the following code at the top of your processing page:

```
<cfif Form.TextBody CONTAINS "%<cf%">
<cfabort showerror="You are very
naughty.">
<cfelse>
 Thanks for your input. Etc…
</cfif>
```

Note that <cfexit> is a better choice than <cfabort> if you're running a custom tag.

Use the poor woman's dedicated SQL server.

You've heard it a hundred times. Put your databases on a different machine. But many maverick developers are unable to afford such a scheme. If you're in this valiant group, there is recourse. At the least make sure your databases aren't in the HTTP tree. If you have no control over this (if you're not hosting yourself), generate a random number to name your database with the following code:

```
<cfoutput>DBCustomers#RandRange
(1,1999999999)#
</cfoutput>
```

If you do have a separate database machine and Web server but one is more powerful than the other, let the faster one be your SQL server. The sine qua non of many ColdFusion applications is a database; therefore, slow database = slow CFServer, but not the other way around.

Portability/Modularity

Avoid proliferating <cfauthenticate> tags since you must specify the security context on the server.

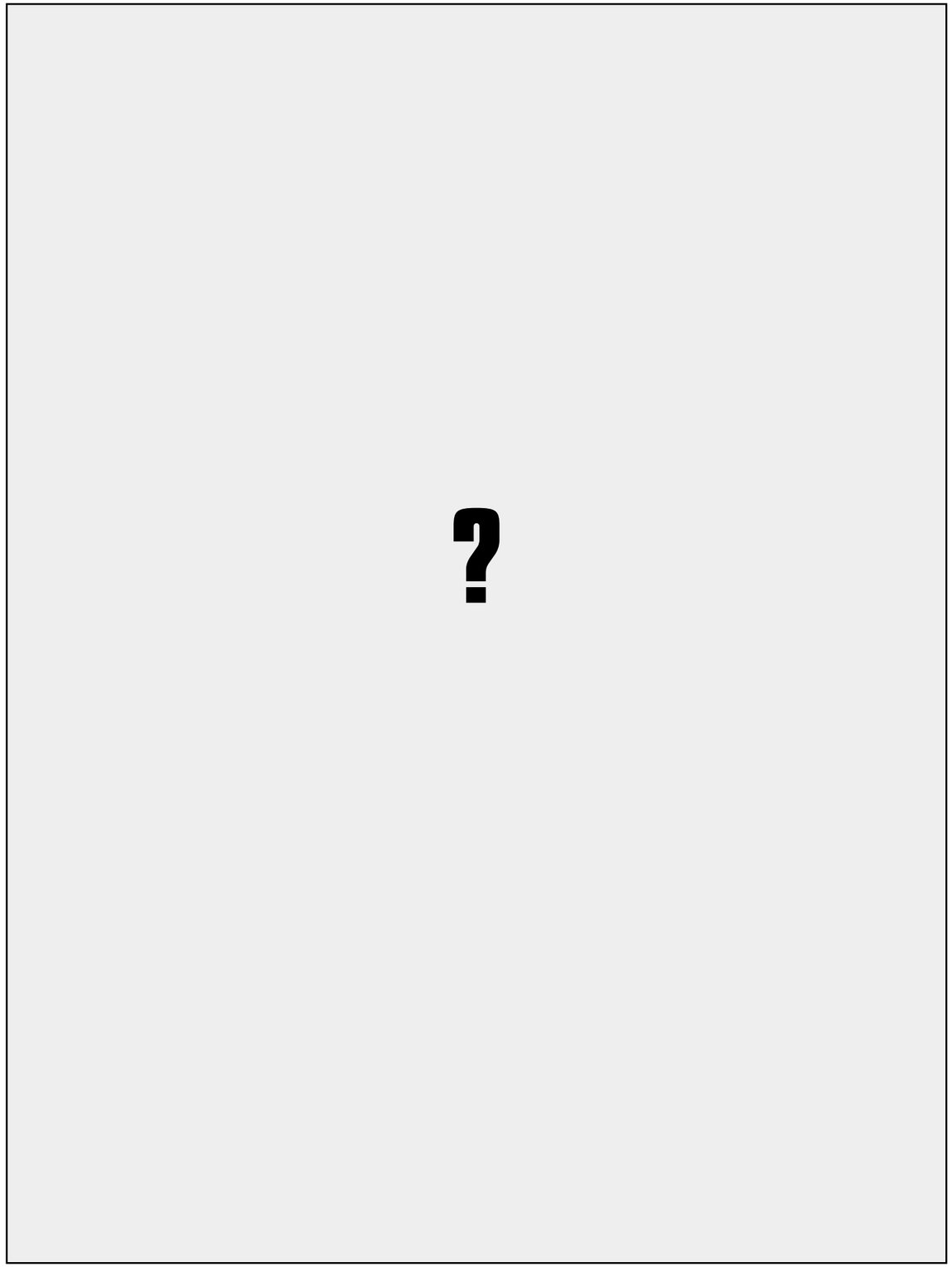
If you move your app to another host or plug the module into an application on a host out of your control or reinstall ColdFusion server or make a significant upgrade, you'll likely have to rewrite your security context, forcing a potentially difficult migration. If you don't face these dilemmas, <cfauthenticate> is an otherwise handy tag.

Format data within your code, not your SQL data type.

For example, imagine you have a field named "Price". Rather than specifying the datatype "money" in your SQL database, specify INT and use the ColdFusion function Dollar format(#Price#). This practice allows you to follow the tradition that dictates one must separate display, action, and content. Also at issue here is portability/migration between database versions and database vendors. This may be more useful to independent developers looking to peddle their application wares than to job shops with their hosting routine down. Note that you'll take a slight performance hit in the (potentially lengthy) interim.

Use relative paths.

In the root of your applications create a folder called "files" or "general" to store single-page items such as



"about" and "contactus" that may not merit an entire folder to themselves. Putting all pages in directories off the root maintains the relative paths for <cfincludes>, images, and Flash menu actions (such as GetURL). If you develop for a number of different clients, it's easy to move an application (or part of one) or reuse the entire app in another site. It makes later additions and changes easier to implement and gives you greater control of definitions and permissions in your Web server.

Speed/Performance

Refrain from running <cfloop>s and other complex logic within your <cfquery> tags.

Because the driver connection remains open for the duration of the code executing within <cfquery> tags, Allaire recommends keeping the executable code therein to a minimum. Anyway, you can accomplish the same task by separating out conditional queries to be run only in the event of a <cfswitch/case> clause as in Fusebox. This reduces to: never hit the database if you don't have to, never ask the database for one column or one row more than you need, and keep your conditional logic in Coldfusion where it belongs.

If database performance is an issue for you, keep this in mind: when designing your tables, accept NULL values sparingly, if at all. SQL Server stores a bitmap in every row to indicate whether a column set to accept NULL values is indeed NULL. By allowing NULLs you force SQL Server to decode this bitmap in each and every called row. This practice multiplies the complexity (read "proneness to bugs") of an application. The server logic em-ployed earns you a second performance penalty as well, as you must account for the allowed values throughout your code.

If you have more than one iteration of the <cfif, cfelseif, and cfelse> tag group, consider using <cfswitch> and <cfcase> to boost performance. In a <cfif/cfelse> scenario, ColdFusion must test against every condition before exiting the clause. If your expression matches the final condition in your series of if/elses, ColdFusion is forced to test and reject every condition. Switch/case is less code text than <cfif/cfelse>, making it easier to follow and debug, and since the conditional logic is more efficient, you can take home a performance boost.

> "When designing your tables, accept NULL values sparingly, if at all

Use nested <cfif> tags sparingly.

I state this as a corollary to the above, though the larger point has been made.

Use the function GetTickCount for testing slow or crashing templates implementing CFX tags or complex CFML logic to isolate the source of the problem.

GetTickcount returns a millisecond counter for timing the processing of ColdFusion markup. Try running the following example code. If this page returns slowly in a no-load environment, you can be confident your performance problem is server side and not due to poor coding or conditional logic. Then test it in a loaded environment, such as the suspect template, where your problem will be exacerbated.

```
<CFSET Count = 1000>
<CFLOOP Index = idxMyIndex From=1>
<CFSET Start = GetTickCount()>
To=#Count#></CFLOOP>
<CFSET Stop = GetTickCount()>
<CFSET RoundTripTime = Stop - Start>
<CFOUTPUT>Round Trip time was
#RoundTripTime#  milliseconds</CFOUTPUT>
```

For Spectra developers, watch this count spiral past 3,000. Then have a heart to heart with your sys admin. (Remember, too, that the IIS 4 Management Console spawns a memory leak if left running. Windows 2000's IIS 5 sports a memory leak so exuberant that it will crash your entire system if you don't run the recently released service pack.)

Trick ColdFusion Studio into quickly saving files in your current project folder, rather than the default Studio folder, for speedy development.

Create a favorite with the following steps:

1. In the Resource window navigate to your project folder.
2. Right-click in the file list.
3. Choose "favorite folders > Add current folder to favorites". Add your folder.
4. Your subsequent effort to File > Save should result in your folder opening first.

Teamwork/Readability

Use lots of comments.

Everyone says this, but the resulting comments from such a vaguely stated rule are vague enough themselves. More important, who hasn't been set back days or weeks by poorly (or un-) commented code they've inherited? Coders must leave breadcrumbs, especially in a landscape where people typically stay at a job for no more than six months. In your application.cfm file, note the following: the author name and e-mail, the date, and a description of the application. As Ben Forta writes in "'The Ten Commandments' – Revisted" (***CFDJ***, Vol. 2, issue 10), "…every conditional statement, every loop, every set of variable assignments, and every include or component reference, must be commented with a simple statement explaining what's being done, and why."

To this I'd add that you note at the top of your list what other processes are dependent on the current process. This way, if I cut anything out of my app elsewhere, you'll be able to tell what else might unexpectedly fall apart. I even go so far as to include a "ReadMe.cfm" file in every application I write. When you comment heavily like this, however, I suggest you get rid of the resulting whitespace. In ColdFusion 4.5 you can do this by going to the Administration Page and enabling "Suppress whitespace by default". Refrain from ending a template with a comment if you choose this setting on the server. On a local level, you can also use the <cfprocessingdirective suppresswhitespace="Yes"> tag to handle it. The downside? You need this tag and its closing counterpart in all the templates you have to remove the spaces.

Use tab stops in your code to make it readable later – for yourself and others.

Validate at the top of the form so you can immediately see which fields contain a validation rule.

Use a standard, conventional nomenclature for your ColdFusion objects.

The Hungarian notation standard used in constructing Visual Basic and C applications offers some direction (see Table 1). So named because it was popularized at Microsoft by Hungarian programmer Charles Simonyi, Hungarian notation is a naming convention that allows the programmer to determine the type and use of an identifier (variable, function, constant, etc.)

ODBC Database	db	dbOrders
Form	frm	frmNewUser
Index object (cfloops)	idx	idxGender
Query	qry	qrySalesByDate
Report	rpt	rptsales
Table object	tbl	tblEmployees

TABLE 1 Hungarian notation

If you don't like Hungarian notation, just make one up with the people on your team – and consider the following:

- ***Mnemonic value:*** Makes it easy to remember the name
- ***Suggestive value:*** So others can easily understand the name
- ***Consistency:*** Similar names for similar objects

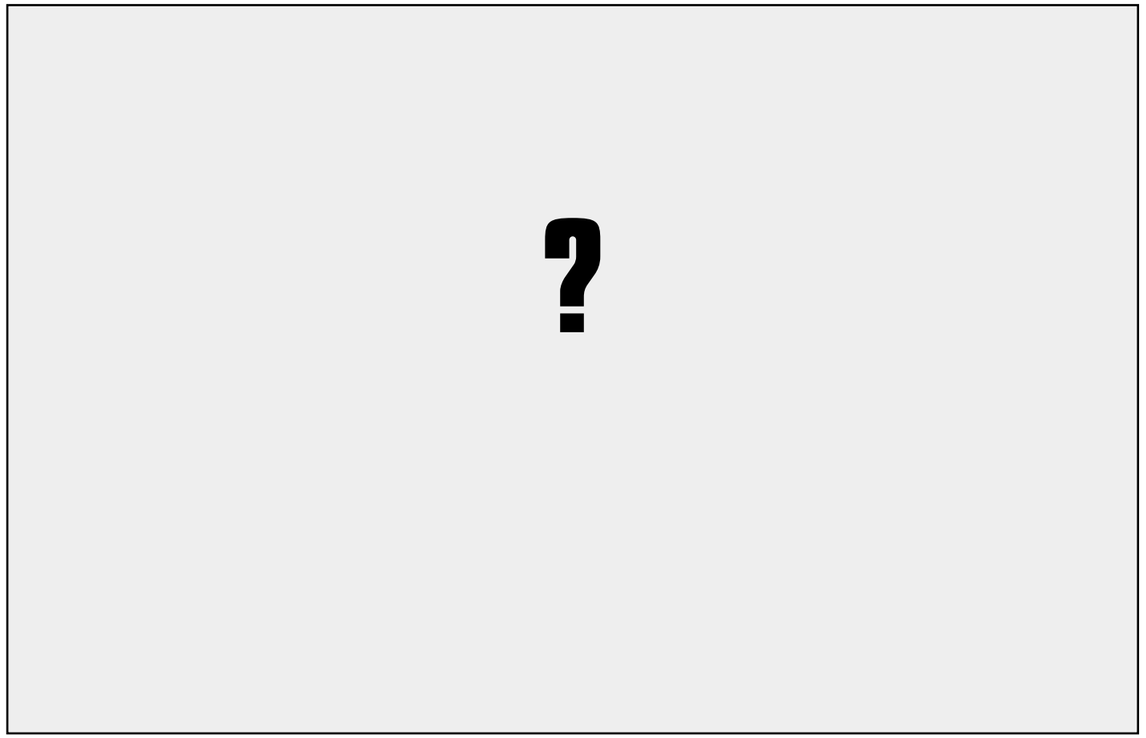
I hope there are one or two suggestions among these that seasoned developers haven't explored and can try out on their own. I also hope beginning coders have found that even if you don't follow any of the suggestions outlined here, it's time saving and aggravation reducing to create a set of your own coding practices that you stick to consistently.

Thank you to everyone who contributed suggestions, especially to Ben Forta for pointing me in good directions for research, and to my friend and colleague Garrick Brooks for contributing excellent ideas on security issues.

ABOUT THE AUTHOR

Eben Hewitt, a certified ColdFusion developer and a certified SQL Administrator, is director of architecture at Canyon WebWorks-Arizona Internet, an Allaire Alliance Consulting and Hosting Partner. He holds a master's degree, which he hopes no one will hold against him.

EBEN@CANYON.NET



Toward Better Error Handling

Part 3: Application-Level Error Handling

BY CHARLES AREHART

When your users experience a CF error, do they still see the traditional default – a plain black-on-white CF error screen? More important, are you being notified when such errors occur? Would you like to log the errors to a database? If you're not taking advantage of such capabilities, you should learn about better error handling techniques that are often missed or misunderstood.

In the first two parts of this series (***CFDJ***, Vol. 2, issues 10 and 12) we learned how to create more effective error handling than the traditional CF default error message. Those articles focused on changes to make in the CF Administrator. In this article we'll focus on implementing error handling choices at the application level.

For those familiar with older forms of application error handling, things have changed dramatically. Many developers still seem unaware of the new capabilities that will dramatically improve your ability to spot and handle errors in your applications.

Site-Wide Error Handling Revisited

Dramatic changes were introduced in ColdFusion release 4.5. Besides what we'll discuss in this article, another new feature in this release was site-wide error handling (discussed in Part 2), which lets you create a new error handling routine that can capture nearly any error occurring in any CF template and process it with complete programmatic control over what should be done with the error.

The benefits are twofold. First, you can format the error to be more attractive, including your own navigational toolbar, color, graphics, font appearance, and more.

Even better, this site-wide error handling routine can perform any ColdFusion tag or function or reference any CF variable, meaning it could send e-mail to the site administrator, store diagnostic information in a database for later analysis, and more. This is a dramatic improvement over what was available prior to release 4.5.

But site-wide error handling, which applies to all applications on a server (site), isn't appropriate if several applications share that single server. Maybe different people should receive e-mail notification of the error, or the data should be stored to different databases, or different applications may want to store different information about errors.

Fortunately, there's an alternative: application-level error handling enabled by the CFERROR tag. But if you think you know about CFERROR, think again. It's also seen a major enhancement in release 4.5.

Older Forms of Error Handling

For those familiar with the CFERROR tag, which has been available since the earliest releases of CF, the news that an error handler can do CFMAIL or database updates may come as a surprise. Until now CFERROR error handling couldn't do either of those things. In fact, error handling templates called by the old form of CFERROR couldn't execute any CF tags at all.

We'll explain CFERROR later for those not familiar with it, but right now I want to clarify for those who did work with it before and were put off by its restrictions, or who indeed wondered why it couldn't be used to automatically send an e-mail when an error occurs, that CFERROR has changed dramatically in release 4.5. New options make it work just like the site-wide error handler described above, including using any CF tag, function, or variable. The change doesn't affect current CFERROR implementations. Instead, it's an opportunity to use the tag in a new form (or maybe drop the older form of CFERROR error handling templates).

Older Forms of Application-Level Error Handling

So what's the CFERROR tag? It's a tag that, like the site-wide error handler feature in the Administrator, lets you describe a template that controls how an error will be displayed and/or handled such that it could present the user with something other than the traditional, simple black-on-white CF error message.

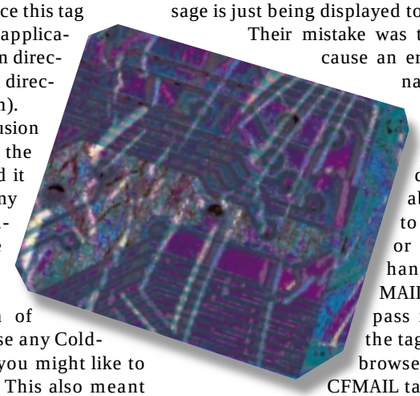
CFERROR TYPE="Request"

The older form of the CFERROR tag had two alternatives, one of which might have appeared:

```
<CFERROR TYPE="request" TEMPLATE="errorhandler.cfm">
```

The "type" attribute meant that you wanted to handle any kind of error in your template (or "request"). You'd place this tag in the application.cfm file that controlled your application, so it affected all programs in that application directory (and any templates in subdirectories of that directory if they didn't have their own application.cfm).

It would indicate that if an error arose, ColdFusion would transfer control to the program named in the TEMPLATE attribute. In our example, we named it "errorhandler.cfm", but you could have given it any name you chose. With control passed to this template, it could be formatted to your preference using any HTML tags to describe the appearance, color, font, and layout of your choosing.



As mentioned, however, this older form of CFERROR (type="request") wasn't allowed to use any ColdFusion tags, functions, or any of the variables you might like to see, including session, URL, form, and cookie. This also meant you couldn't leverage any previously written code using CFINCLUDE, which meant you had to hard-code any headers, footers, or navigational toolbars you might want to present to the user. And, of course, you couldn't do a CFMAIL, CFQUERY, or any other tag.

You may wonder why this was at all useful, and in fact many people felt the same way and never bothered with it. It did have one means to be a little more useful. As I discussed in the last installment on site-wide error handlers, ColdFusion exposes a set of "error variables" for use in any error handling templates, including that pointed to by CFERROR. They include:

- ***Error.DateTime:*** Date and time of the error
- ***Error.Template:*** Template in which the error occurred
- ***Error.Diagnostics:*** Actual CF error message
- ***Error.HTTPReferer:*** The page that called the template getting the error
- ***Error.QueryString:*** The URL string, if any, passed after the name of the template (the raw format of any "URL" variables available to the page)
- ***Error.Browser:*** Type of browser calling the page in error
- ***Error.RemoteAddress:*** IP address of the user calling the page in error
- ***Error.MailTo:*** Mailing address designated to receive e-mails sent by the user choosing to respond to the error message

The page could be formatted to display those variables. If you think about it, though, how could you output such variables if you can't use a CFOUTPUT? Well, Allaire got around that by letting you specify those variables surrounded by just pound signs and with no CFOUTPUT needed. You still couldn't refer to any other variables, though.

It was a step above the traditional, plain CF error message, because it could have a more consistent look and feel with the rest of your application, but it was of marginal use in creating a complete error handling mechanism. What developers really wanted was a means to be notified when an error arose.

The Misunderstood MAILTO Attribute

Many developers interested in the capability of receiving an e-mail notification about an error were excited to see the CFERROR tag supported a MAILTO attribute. You could write the tag as:

```
<CFERROR TYPE="request" TEMPLATE="errorhandler.cfm"
MAILTO="admin@youhost.com">
```

Sadly, it didn't accomplish what most were expecting. The Allaire forums are littered with folks lamenting that "CFERROR's not working. I'm not getting any e-mail, and indeed the mail message is just being displayed to the user when an error occurs."

Their mistake was that they expected this attribute to cause an error to be sent automatically to the named administrator. Instead, it simply populated the Error.Mailto variable mentioned in the previous section.

So why was the e-mail just being displayed on the screen? Well, think about it: error handlers aren't allowed to use (and won't execute) any CF tags or regular CF variables. If an error handler mistakenly tried to do a CFMAIL, CF would ignore it and simply pass it and the mail message between the tags to the user as HTML. Of course, a browser doesn't know what to do with a CFMAIL tag either, so the user simply saw the mail message displayed to them. Not what was expected at all!

The real intent of the MAILTO variable was to display, as part of the error message you formatted, a hyperlink the user could click on to send an e-mail to the named address (using the mailto: protocol, as in <href="mailto:#error.mailto#">Please notify the administrator of this error).

Of course, you had to hope they'd bother to take that link and that they'd be willing to copy and paste the error on the browser screen into the e-mail. Even then, this e-mail couldn't really capture everything you needed to know about the error, since you didn't have access to all the CF variables in play at the time of the error (form, session, local, etc.).

Circumventing the Limitations: Old Tricks

Getting notification about the error is of course a critical need. The MAILTO attribute alone and presenting a mailto hyperlink to

the user just didn't cut it. Clever CF programmers realized there were several ways to get around the e-mail limitation.

One approach was to create a form in the error handler template in which you stored all the error variables in hidden fields on the form. The user would be asked to submit the form, passing control to a real CF template, which could then send an e-mail or create some sort of error database log entry.

A natural extension of that, for those with JavaScript skills, was to simply have that form submitted automatically by way of a JavaScript submit() method. An example of this is presented in the Allaire Forums at http://forums.allaire.com/devconf/Thread.cfm?&&MessageID=178597&#Message 308262.

Fortunately, the newer form of CFERROR can get around the need to play those games and also provides a lot more about the state of the program at the time of the error. Now you can simply do a CFMAIL and much more in the error handler itself. Before we get to that, let's look at the other of the two older forms of CFERROR processing.

CFERROR TYPE="Validation"

There were actually two older forms of CFERROR, the other being TYPE="Validation". This feature worked (and still does) much the same as TYPE="Request". If you created a form that used the "hidden field" validation enabled by ColdFusion (_required, _date, _integer, etc.), this form of CFERROR could name another template that could lay out the appearance of the messages shown when a user's input failed to meet the validation. There are three special "error" variables for this handler:

- ***Error.InvalidFields:*** Detailed validation error messages
- ***Error.ValidationHeader:*** The wording that normally appeared at the top of such a form validation error message display
- ***Error.ValidationFooter:*** The wording that normally appeared at the bottom of such a form validation error message display

Each of these older forms of CFERROR, while interesting and useful in at least providing a more consistent interface when errors arose, was limited and not at all capable of providing you, as administrator of the application, with an e-mail to know that an error had occurred, let alone a means to store a database to track such errors.

Newer Forms of Application-Level Error Handling

Allaire realized that people wanted an error handler to do more. So they added new features in release 4.0 (CFTRY/CFCATCH) and release 4.5 in the form of site-wide error handling and two new types of CFERROR handling, each of which have complete access to all CF tags, functions, and variables. Finally, we're able to create full-powered error handlers.

> "You can now take note of the session, application, client, cookie, cgi, form, URL, and other variables in play at the time of the error

A Brand New CFERROR TYPE="Exception"

The first new form is TYPE="Exception". On a simple level this is just a new "type" of CFERROR error handler, and the format could be specified as:

```
<CFERROR TYPE="Exception"
TEMPLATE="enhanced_errorhandler.cfm">
```

Since this template, which we've called enhanced_errorhandler.cfm, can do any tags, functions, or variables, we can have it do a CFMAIL (using the MAILTO attribute, if specified, or any other CF variable or hard-coded value) or a CFQUERY to insert a log database record, and more.

The ability to access any variables shouldn't be underestimated. You can now take note of the session, application, client, cookie, cgi, form, URL, and other variables in play at the time of the error. This makes a tremendous difference in being able to diagnose the cause of an error. In Part 2 of this series, "Site-Wide Error Handling," I offered some code samples to allow you to loop through all the variables in each of these scopes. Wonder how you can have access to the form and URL variables? If a CFERROR transfers control to the error handler, doesn't that lose the form and URL variables? Nope. It appears to work more like a CFINCLUDE, so those variables are there if they were passed to the page that had the error.

More on the Error Handling Hierarchy

Note that you can certainly have both the old and new forms of CFERROR in the form of two different tags in the application.cfm. There's no significance to the order in which they're specified. The real significance is that the "request" handler, if specified, will handle any errors the "exception" handler doesn't.

When would it step in? How about if there's an error in the "exception" handler? This often happens when people are getting started. If they have an error in the "exception" handler, control passes to the older style "request" handler and no automatic e-mail or database log is created. The problem is simply an error in the handler, but it's hard to diagnose at first, especially if the output to the screen for the two error handlers is the same.

What if there's no "request" handler when an error occurs in the "exception" error handler? Then CF will look for a site-wide error handler, and if there's none, the old-style, plain CF error message will be displayed.

Speaking of the hierarchy, it's more accurate to say that the exception handler will handle an exception that hasn't otherwise been handled. As we'll learn in the next installment, you can handle errors at an even finer level of control, within a piece of program code, with the CFTRY/CFCATCH tags.

The Curious Type="Monitor"

I mentioned earlier that there were two new types of CFERROR mechanisms. The second is Type="Monitor". It's one that's often misunderstood and seems relatively poorly documented. It's like the new "exception" type in its power to use any tags, functions, or variables. So what's the difference? It's a subtle one, and it will make a lot more sense if we wait until after we learn about CFTRY/CFCATCH and exception types in the final installment.

Problems Using the New Error Handlers

Beware the documentation and old articles, forum notes, and so on.

The first thing to be aware of as you try to learn more about these new forms of error handlers is that many resources you look to may have dated or incomplete information. They're still relatively new, and the resource (Allaire forums, articles, and so on) may be referring to what was true when that resource was created.

Even then you may find that conversations with other developers lead to a variety of explanations of what can and can't be done. I don't claim to have written the authoritative explanation (far from it), but I still hear from otherwise guru-level people who don't quite get all the subtleties of these new forms of error handling.

Indeed, even the Allaire documentation shows occasional lapses that suggest they've only been updated to mention the new features. The problem is that some comments remain that are relevant only to the "old style" of CFERROR processing. For instance, the CFERROR usage topic in the reference manual states:

"To help ensure that error pages display successfully, pages you specify with CFERROR should not be encoded with the cfencode utility."

This was true for the older style CFERROR types; while they were .cfms, they weren't processed as such, so an encoded file wouldn't display properly to the user. It's not an issue at all for the two newer types of error handlers (or indeed the site-wide handler). It's not that big a deal, but it's reflective of a concern you should have about the details you may read about on this subject.

Problems Using Code in Earlier Versions

It's important to keep in mind that not only do these two new CFERROR types (exception and monitor) work only in 4.5 or beyond, they'll also specifically fail if used in an older release. It's not enough to test for the version (using the variable Server.ColdFusion.ProductVersion) and execute the code only if running in 4.5. If the version is not 4.5 or above, and you try to use, for instance, CFERROR TYPE = "exception," you'll get a compilation error.

What if you have to move code back and forth between a server that's 4.5 or greater and one that's not? Here's a trick:

```
<CFSET Version= ListGetAt(Server.ColdFusion.ProductVersion,1, ", ")
& "." & ListGetAt(Server.ColdFusion.ProductVersion,2, ", ")>

<CFIF Version GE 4.5>
 <CFSET Type="Exception">
 <CFERROR Template="error_handler_for_exception_type.cfm" Type="#type#">
</CFIF>
```

Handling 404 Errors

It's also important to note that these error handlers can't capture "404 (file not found)" errors caused when a user types in a URL for a file that doesn't exist. If the file they're requesting is a .cfm file, there's a solution: a new "missing template handler" in the admin, which allows you to customize the error appearance. If the file they're requesting is an .htm or other file, you'll need to set up your Web server to handle such missing files.

Summary

There you have it. I hope this quick tour of application-level error handling has been informative and proves useful. With this knowledge, no site or application should ever again show the user the plain black-on-white error message!

The final installment of this series will examine the CFTRY and CFCATCH approach to error handling. We'll show some capabilities enabled by that, such as detecting and resolving database errors in a more well-planned manner. "Code-level" error handling gives you a finer level of control than application-level error handling. Until then, you can learn more in the Allaire docs as well as in ***CFDJ*** articles, such as "What's In It for Me? Taking Advantage of New Features in ColdFusion 4.0," by Michael J. Murdy at www.sys-con.com/cold-fusion/archives/0102/murdy/index.html.

Finally, there are several other topics on using CFERROR that we've run out of time to cover. You can see an extended discussion on my Web site at www.systemanage.com/articles/.

About the Author

Charles Arehart is a certified Allaire trainer and CTO of SysteManage, an Allaire partner. He contributes to several CF resources and is a frequent speaker at user groups throughout the country.

carehart@systemanage.com

?

Worst Practices

A top 10 list for CF and Fusebox

BY HAL HELMS

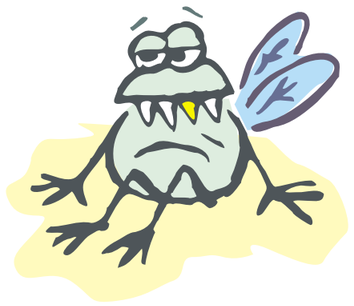
"If debugging is the art of taking bugs out of programs, programming must be the art of putting them in." —Anonymous

I've been hard at work creating material for a "Best Practices with ColdFusion & Fusebox" training class and it got me thinking: Why don't we ever see a worst practices class? Heaven knows there are some real stinkers out there – why aren't they shared with others, so that all may learn? If something's worth doing badly, isn't it worth doing really badly? Or, as one of the regulars on the Fusebox list notes in his signature line: amateurs built the Ark; professionals built the Titanic.

Yet, as I look about, all I see are people working on improving code. Go to Figleaf's site, for example (www.figleaf.com). What do you find? Lots of stuff made to help coders become better coders: code samples, presentations, and more.

Or go to www.fusebox.org – same thing – lots of stuff to help people write better code. My friend, Adam Churvis, is practically giving products away to help developers at www.commerceblocks.com.

And, loath as I am to confess it, even I am not blameless; www.halhelms.com has tutorials and even free online classes all designed to – you guessed it – help developers write better code.



Now writing good code is fine – if that's what you want to do. However, from looking at some code, I'm not convinced that's what people want to do. For example, here's a bit of code that, I swear, came from a production environment:

```
<cfloop
 from="1"

to="#ListLen(Trim(ValueList(myQuery.Name)))#"
 index="i">
#ListGetAt(ValueList(myQ.Name),i)#<br

>
</cfloop>
```

Can anyone argue that the coder was trying to write good code? I think not. The programmer looking to write good code would probably have chosen, instead, something more mundane:

```
<cfoutput query="myQ">
 #name#<br>
</cfoutput>
```

I think you'd agree the first snippet is far more…far more…how shall I put it?

Bad. In fact, it's so bad, so amazingly, needlessly complex, that for it to lie in obscurity forgotten seems terribly wrong. Surely if anything does, this code calls out for a Worst Practices award.

I've found that there are three main reactions to code such as this. The first and most common reaction is despair. "I can never aspire to this level of incompetence," you say. That's a natural reaction. But I'm here to tell you: you really can be that bad – but you'll need some help. Help that I never received. In looking over some of my own early code, I can see in retrospect that I showed real promise. I can only wonder how far I might have gone had this early flicker of potential not been snuffed out by exposure to really good programmers.

The next reaction to the code sample above is cockiness. "I could write bad code like that standing on my head in a snowstorm." No, you could not, and remember, Worst Practices code is not just "bad code." Although a proper analysis must be left to another time, suffice it to say that just as taking scissors to a normal painting and rearranging the resulting parts won't yield a Picasso, neither will ignoring best practices produce true Worst Practices code. Although it's a start.

The final reaction comes from the person who says, "But of what use is Worst Practices code?" And I say, "Of what use is a mountain, or a sunset?" Some things are just for their own sake.

So in the hopes that you fickle readers will remember this community service when next year's ***CFDJ*** awards roll around, I offer you this free (for now) Top Ten List of Worst Practices for your nonedification.

1. *Using creative naming schemes*

Anyone determined to master Worst Practices code will start here. The would-be Worst Practices programmer will find all sorts of ways to make his or her code obscure and unreadable. And maintainability? Hello?

Here are some examples:

- ***variable_with_liberal_use_of_underscores:*** Much to be desired for the way it causes the programmer to use a two-key combination to create the underscore. And since most programmers can't find the underscore key by touch, they have to stop and look at the keyboard. A nice touch.
- ***pro_id:*** Note here the abbreviation pro. Does this stand for product? Project? Program? Something else entirely? Who knows – certainly not the person who has to maintain this code. Score one for Worst Practices and use abbreviations liberally in your code.
- ***Equals as a variable name:*** For this inspired bit of advice I must give credit to Roedy Green (whose brilliant site at www.mindprod.com/unmain.html provides tips that should be studied by all aspirants to the Worst Practices Hall of Fame). Imagine the code you can write using equals.

```
<cfif equals GT 10>
```

That should keep them guessing for a while.

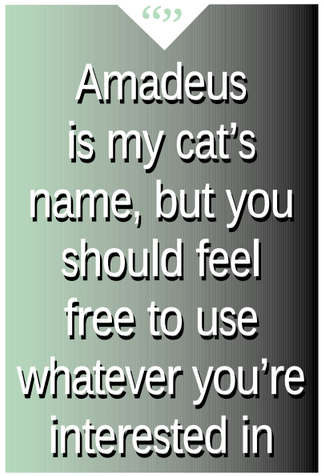
- ***amadeus:*** Yes, this old chestnut should not be forgotten. Use whimsical names for variables in order to keep the "fun" in programming. Amadeus is my cat's name, but you should feel free to use whatever you're interested in.

There are so many more, but I must move on to…

2. *Commenting your code*

Of course, it's best not to comment your code at all, but you may find that due to political pressures, you must provide some level of comments. Still that doesn't mean you have to like it, so why not let your displeasure show in the way you write comments? Here's some code I found….

```
<!--- Loop over the fields in the
form. If any of them match a cookie
that was set previously, user wants
to make a substitution. This will go
in "substitutions" that will be
passed on.  --->
<cfloop list="#form.fieldnames#"
index="aField">
 <cfif IsDefined( 'cookies.' & i  )>
  <cfset substitutions = ListAppend(
substitutions, aField )>
 </cfif>
</cfloop>
```



Note how the comments tell why the code executes rather than how it executes? This is the hallmark of good comments. What can I say? This coder just doesn't "get it." Now look at my reworked Worst Practices code:

```
<!--- Loop over list. If a list item
matches a cookie, put list item in
another list. --->
<cfset amadeus = form.fieldnames>
<cfloop list="#amadeus#" index="i">
 <cfif IsDefined( 'cookies.' & i  )>
  <cfset ludwig = ListAppend( ludwig,
i )>
 </cfif>
</cfloop>
```

All the comments are entirely trivial. They indicate nothing that a cursory glance at the code won't reveal, while the meaning of the code is impossible to discern. Admittedly, it isn't easy to get to this level of obscurity, but it's certainly something to strive for.

In Louisiana they use the word *lagniappe* to describe "a little something extra." There's a lagniappe in the converted code. Do you see it? It's the use of i as the variable name for an index. This was originally used in languages whose loops only gave you a handle on the integer value of each element as it was being looped over. You started with 0 and then worked your way through the number of times looped. But with ColdFusion you have a handle on the actual value in the list itself. You can see in the original the misguided coder uses the term *aField* to describe…well, a field. I think you'll agree my code is far less clear.

3. *Duplicate functionality*

If something's worth writing, it's worth writing twice. Or three times. Or more. That's the motto of the Worst Practices coder.

Here's some "unimproved code":

```
<cfmodule
 template="GetStateTax.cfm"
 state="#form.state#"
 total="#subTotal#"
 returnValue = "stateTax">
<cfset total = subTotal + stateTax +
shipping>
```

The coder created a module to return the state tax for any state. This module can then be used wherever the application needs it. It can also be used by other applications as it's generic. But all those words! Look how much "simpler" the Worst Practices code is:

```
<cfset total = subTotal + subTo-
tal*0.85 + shipping>
```

Now, when the state tax rate changes, you'll have to search through every file that might be affected. Using the unimproved version, you'd only have to make a change to a single file.

You are taking notes, aren't you? Okay, then let's move on to…

4. *Don't use third-party code*!

The Allaire Developer's Exchange (http://devex.allaire.com/developer/gallery/) is full of code that others have sweated over and debugged. Do you honestly think a Worst Practices coder would use such a thing? Please…

Let's say you need to sort a two-dimensional array, something that ColdFusion has no built-in function for. If you immediately thought, "I'll bet there's a tag for that!" slap yourself! Of course someone has already written a tag for it. What's that got to do with anything? Surely if you're willing to write your own code twice (see Tip 3), you should be willing to duplicate code someone else has written. I hope that's clear.

5. *Use global variables – a lot!*

One of the more important Worst Practices is using global variables liberally. You never know when you might need a piece of information. Best to make it global! At first glance it might not appear self-evident why this should be a Worst Practice, but consider that as a program's number of moving pieces increases, global variables are like landmines, hidden from view. There's no telling when someone paying attention to this tip will accidentally overwrite one that you wrote – and then the fun begins in earnest!

6. *Don't use assertions to test variables*

There's a whole class of errors that occur because code is expecting a variable of a certain type or its value lies within certain bounds. Consider this code:

```
Thanks to our FriendlyShopper(tm),
your shipping works out to only
$#Val( shippingCost/packageQuantity
)# per package. Such a deal!
```

This works fine until you get a packageQuantity of zero, then your user receives a little lesson in the impossibilities of division by zero. The ColdFusion tag, <cfparam>, offers some rudimentary protection against such types of errors.

Recently I wrote a column on assertions (as such error protections are often called) in ***CFDJ*** (Vol. 2, issue 10) that provided a custom tag for implementing assertions in ColdFusion. Avoid it.

7. *Error handling is for others*

Exception and error handling is based on the notion that something unexpected can occur. But is this notion really true? To our code?

Still not convinced? Tell you what, let's do it "later," okay?

8. *Testing is for losers*

I'm sorry to be so harsh here, but sometimes the cold truth is called for. Testing code is an implicit admission that your code might not measure up. Is that the message you want to give to others, that you're not good enough?

When people report "anomalies" in the code, the savvy Worst Practices coder demands to know every detail about the alleged "error," including:

- Time of day
- Exactly what were the last 100 keystrokes that were pressed immediately prior to the condition (a much better word)
- What operating system the user was running. Were all service packs installed? When were they installed?
- Other applications on the user's disk (whether running or not at the time)
- Whether AOL was involved
- Whether any aurora borealis activity was present in the user's region
- Whether the user can spell aurora borealis

> "One way to eliminate errors is to make dental surgery preferable to reporting them

You get the point, I think. Some wise man once pointed out that one way to reduce the crime rate is to make theft legal. I call this "thinking outside the box." It's also true that one way to eliminate errors is to make dental surgery preferable to reporting them. No reports, no errors. Sometimes, simple is better.

9. *Avoid JavaScript for form validation*

Despite Selene Bainum's excellent article in the August 2000 issue of ***CFDJ*** (Vol. 2, issue 8), explaining how simple it is to use JavaScript and <cfform> together, Worst Practices coders are adamant about avoiding JavaScript for form validation. It smacks too much of assertions and code testing.

If you can't avoid form validation entirely, at least do it on the server with ColdFusion. That way the user will have to wait for the round-trip to the server and back. Consider it a very gentle rap on the user's knuckles. Maybe next time they'll be a little more careful and figure out that when you asked for their phone number, you wanted it in the format 555-555-5555, not (555) 555-5555.

Granted, you didn't tell them this, but a Worst Practices coder isn't really into that touchy-feely user-interactivity thing. Leave that for the losers who test their code.

10. *Mentor others*

Worst Practices coders know that they must "give back" to the Worst Practices community. They do so by teaching others these techniques and thus help to swell the ranks of Worst Practices coders. But, if I may wax philosophical for a moment, tips like these aren't enough. Coders – especially junior coders – are eager to do good work. They're likely to want to do such things as attend classes, read books, and discuss new ideas and new ways of doing things.

In this context "mentoring others" means crushing all enthusiasm by applying generous doses of sarcasm, stony silence, and implications that "real coders" don't worry about/need that stuff. Worst Practices coders remember how "helpful" their bosses were, and once they get into a position of power, they follow the Biblical injunction to "go and do likewise."

Sadly, our tour of Worst Practices must end. I'd like to close with a quote from the great American cowboy–philosopher, Will Rogers, who might well have been talking about this very subject when he said, "If stupidity got us into this mess, then why can't it get us out?" I'd like to think that this column has pointed a way in that very direction.

ABOUT THE AUTHOR

Hal Helms (www.halhelms.com) is a Team Allaire member who provides both on-site and remote training in ColdFusion and Fusebox.

HAL.HELMS@TEAMALLAIRE.COM

?

WAP **Revisited**

BY
BEN FORTA

The challenges of wireless development

Fourteen columns ago in ***CFDJ*** (Vol. 1, issue 6) I wrote about wireless computing having finally come of age, the importance of WAP, and that ColdFusion developers could leverage this new and exciting technology quickly and easily.

During that time WAP has become quite the buzzword. Developers seem to love it, industry pundits love to hate it, and all the while more and more WAP devices and content are becoming available. According to Pinpoint.com there were less than 1,000 publicly available WAP pages at the end of 1999 and over 2,000,000 by June 2000 (a growth curve that even dwarfs the Web's). And based on current trends, IDC is predicting that by mid-2002 more users will be accessing Internet services using wireless devices than traditional wired connections (this is already the case in Japan).

What all this means is: like it or not, WAP is here to stay. With WAP 2 on the horizon (a more complete specification that adopts XHTML instead of the more "proprietary" WML), WAP creates interesting new possibilities and opportunities for developers willing to give it a try.

In this month's column I'd like to revisit WAP, but not from a beginner's perspective (for that, refer to my aforementioned column). Rather (based on the questions I get asked most frequently), I'd like to take a big step back and look at some overall design and development considerations – things to keep in mind when looking at WAP.

The "Killer App" that Isn't

I often get asked, "What is the killer app that will make WAP successful?" Others get asked this too, and I have to disagree with almost every response I've seen thus far.

There is no killer WAP app – the platform and devices are not designed for applications, never mind killer ones. If users wanted to run apps, they'd use browsers or more traditional clients. WAP is used because it allows instant access to small, bite-sized chunks of highly relevant information. The relationship of WAP to the Web is similar to the one between pagers and cell phones. Your phone is used for full-blown conversations, while pagers are used for small bursts of immediately needed information. Even though most phones nowadays are pagers too, the analogy still works – one does much less but is well suited for short bursts of information; the other does much more but has more overhead. The same is true for WAP and the Web.

> "There is no killer WAP app – the platform and devices are not designed for applications, never mind killer ones

The "killer app" is actually not an app at all – it's data, any data that's time-sensitive. Examples are:

- Flight time and gate changes (I rely on this one extensively)
- Stock price alerts (notification and perhaps even the ability to take action)
- Phone number lookups (with the ability to dial the found number)
- Maps and directions

The common denominators here are that all this data is finite in scope, highly relevant when needed, and requires minimal user interaction. That's the kind of access that makes WAP compelling.

Incidentally, many columnists (including some very respected ones) have denounced WAP with a "who would want to surf the Web on a device with a postage stamp–sized screen" argument. And they're right, who would? The problem is, they're totally missing the point here – the fact that they're thinking about surfing the Web means they've completely misunderstood the use of this technology. They're right, no one should be surfing the Web on a WAP device, but for instant access to the data you need when you need it, WAP makes all the sense in the world.

Forget Pretty, Make It Functional

Have you ever compared today's Web sites with those of a few years ago? There's no comparison. Simple text and graphics with basic formatting have given way to sophisticated (and often heavy) user interfaces that closely resemble complete multimedia applications more than anything else. Users' expectations have changed, and sites have to look good to attract an audience. So developers go to great lengths to ensure that content looks perfect on all browsers and on as many platforms as possible. Nowadays functionality isn't enough on the Web; it needs to look good too.

This is proving to be one of the most difficult transitions for Web developers looking at WAP – in the WAP world functionality is not the primary objective, it's the only real objective. Making screens look pretty is pointless for a whole slew of reasons:

- Device capabilities differ so dramatically that even basic text formatting doesn't work consistently across them all.
- Screen size is usually very limited.

?

If you had standard logos, headers, footers, and formatting, there wouldn't be any space for content.
- Connection speeds are slow, and you can't afford to multiply the amount of data to be downloaded just to make things look nicer.
- Most important, users don't want pretty screens, they want content – plain and simple. If they wanted a fun surfing experience, they'd be using a Web browser. As explained above, WAP (and other wireless technologies too for that matter) is used primarily for instant access to relevant information, and that's it.

In other words, stop thinking about screen layout and content formatting – for the most part you can't control it anyway. Concentrate on presenting the content users want in as simple and intuitive a fashion as possible. That's priority number one.

Keep Bandwidth in Mind

This brings us to connection speeds – although I'd rarely use the word "speed" in conjunction with WAP. Not only is connection speed limited, the packet size is too (and terribly inconsistent from one device to the next). Phone.com browsers support between 1,492 to 2,000 bytes per packet, some Nokia devices support smaller packets, and some Ericsson devices support larger packets. In addition, network latency rears its ugly head even at packets of this size, so as a rule packet size should be kept at around 500 bytes (not very large at all).

Before you go off to check file sizes, it gets a little more complex. Data is compiled when sent to the device (instead of as raw text), and the packet size you need to watch is the compiled packet size (which can be bigger or smaller than the raw text size). You usually don't have to compile data yourself (the WAP gateway does that); however, even though you're delivering raw (uncompiled) text, you should still know the number of bytes it'll compile to. Most development SDKs (like Phone.com's UP.SDK and Nokia's SDK) feature mechanisms that determine compiled content size. These options should be used.

Another important note is that every effort should be made to fill packets – it takes just as long to send a packet that's half full as it does to send a packet in which every byte is used. When planning your design pay attention to this – perhaps you could send multiple cards (a deck) instead of a single card so the next request won't need a round-trip to the server, or perhaps you could afford to provide more information within a card (preventing the need for another request) if space permits.

Bandwidth concerns will go away eventually, but you can't afford to wait until then. If you're serious about WAP development, you need to worry about this right now.

Emulators – Love Them and Hate Them

I've mentioned that the SDK is available from Phone.com and Nokia. These SDKs include device emulators – software programs that let you simulate a device's operation on your computer. Emulators are important, and you must use them. Actually, I'd go as far as to say that you must use them all, or at least the major ones:
- Phone.com
- Nokia
- Ericsson
- Motorola

Why use them all? Because the more platforms you test your code on, the better, and it's more likely your code will work when released into the wild.

But don't rely on emulators – ever. They're called emulators for a reason – they're not real devices, they just emulate (pretend to be) them. Emulators can give you a good idea as to how your code will behave (and the better emulators actually run the same software that's running on the device being emulated), but there's no substitute for real-world testing. In an ideal world you wouldn't need emulators at all, but this is not an ideal world. Developers don't have access to all major devices. Some devices operate only in specific parts of this planet, and the exact same device can function differently based on the provider using it. So emulators are needed, but they must be viewed for what they are – emulators. Before you deploy your app you must test it on real devices, as many as you can. You should even try to find users with other devices and enlist them in the testing, even users in other countries. This should be obvious, but I'll say it anyway – the more you test, the better.

Rethink User Interfaces

No pointing device, no keyboard, limited graphics support, no DHTML, limited client-side scripting, no colors, no CSS, no frames or layers, no sophisticated form controls, limited text and font control – the list goes on and on. Effective WAP user–interface design is a challenge, one that requires us to rethink user interaction. Here are some points to consider:
- Don't publish too much content, as most of it won't be readable anyway. The key is to be focused: a focused array of options and focused content within each.
- Use menus for everything since selecting options is very easy on most devices, but don't overuse them. If users have to drill down through umpteen levels of menus, they won't use your app.
- Frequently used menu options should always be listed first.
- The less data entry, the better. Some newer devices are simplifying data entry (with different input types, T9, and other technologies) but even so, data entry is a painful experience at best.
- Wizard-style interfaces work really well, but try to keep as many cards as possible in a single deck so as to prevent lots of round-trips to the server in the middle of a data-entry sequence.
- Request data intelligently. For example, if you need a U.S. city, state, and zip, just ask for zip and work the rest out yourself.
- Make user-provided data persist. Any time you can remember data so users don't have to retype it, the better.
- Always provide a backward navigation option, and not necessarily to the previous card. Allow users to easily get "back" to strategic locations within your application.
- Resist the urge to keep shouting out your company name, logo, and more. Nothing annoys WAP users more than having to scroll down to see content because a company name and logo took up all the visible screen space.
- Creating device-specific content (based on detected capabilities) is extremely complex and highly error-prone. It's great if you can do it, but only do it if you're going to invest significant time and resources in the effort (and testing). If you can't make that investment, don't do it at all and just stick to basic (common) functionality.

Writing for HTML and WAP

I know I'm going to get in trouble for this one (I've already received hate mail for saying this in my book, *WAP Development with WML and WMLScript*), but here goes – don't even bother porting existing HTML applications to WAP, and for that matter, don't try writing code that supports both.

As explained above, WAP sites and Web sites have different goals. If you're trying to port an HTML app to WAP, I suspect you're not creating a WAP site that will truly deliver what

WAP users want. What makes a good Web site is not what makes a good WAP site, and the reverse is true too. The goals are different, the user experience is different, the navigation is different – there's usually more that's different than the same.

Even if the HTML and WAP sites were so similar that you could use XML and XSL to render the content in different ways, chances are it still won't work properly. For example, a single HTML page doesn't map to a single WAP card – sometimes a single HTML page needs to be multiple cards, other times many HTML pages need to be a single deck of cards. Trying to render content for both platforms at once usually means that you deliver content for neither effectively.

If you're serious about WAP (and I believe you should be), you need to take a big step back when thinking through your wireless application. If you can reuse existing code or content, great; if not, don't try and force it anyway. It just won't work.

Having said that, I do believe content can (and should) be reused. If you've created your Web application correctly, separating content from presentation and business logic, you'll likely find that content (and perhaps even business logic) can be reused. (Incidentally, this is why Spectra is so well suited for HTML+WAP development). If you're developing code from scratch, you should definitely tier your code this way in order to reuse as much of your investment as possible.

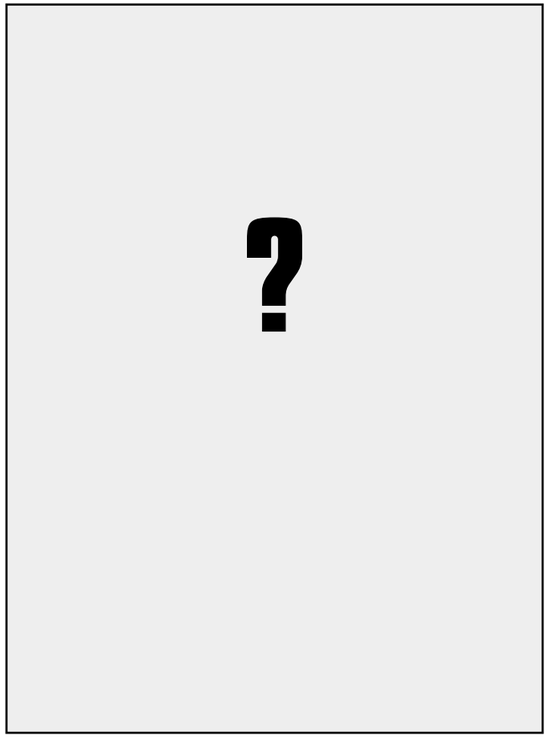
Summary

WAP is an exciting technology, and one that has great potential – if used properly and where appropriate. If you put the effort into designing your applications correctly, getting to WAP can be quite painless (and fun) too.

ABOUT THE AUTHOR

Ben Forta is Allaire Corporation's product evangelist for the ColdFusion product line. He is the author of the best-selling ColdFusion 4.0 Web Application Construction Kit *and its sequel,* Advanced ColdFusion 4.0 Development, *as well as* Allaire Spectra E-Business Construction Kit *and* Sams Teach Yourself SQL in 10 Minutes. *He recently released* WAP Development with WML and WMLScript.

BEN@FORTA.COM



A Cold Cup **o'Joe** Part 2 of 8
Working with Servlets

Playing nice and making complements

BY **GUY RISH**

One of the most important lessons to be learned with any Internet technology is to play nicely with others. In the ever-expanding horizons of ColdFusion this means Java and in the scope of this article, servlets.

There are times when ColdFusion and servlets seem to overlap, and people have a tendency to see them as competing product technologies. There are, however, numerous areas where they can act in a complementary fashion, seeming less like warring camps and more like cooperative agents.

Servlets can provide access to the wide array of Java libraries, which can be used to enhance or extend ColdFusion's capacities with such things as string manipulation, XML manipulation, networking, compression, encryption, graph generation, image creation and manipulation, and PDF creation or formatting. On the Web, both the Allaire Developers' Exchange and Sun Microsystems' servlet resource page (http://java.sun.com/products/servlet/resources.html) provide numerous useful and free servlets.

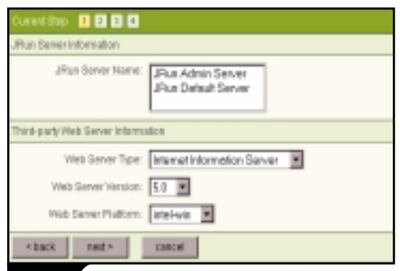


FIGURE 1: External Web Server Connector Wizard, Step 1

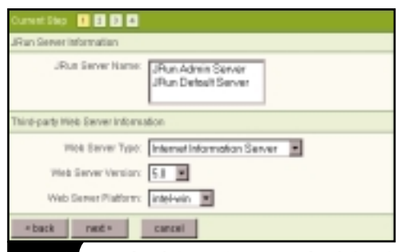


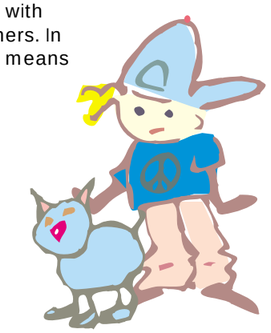
FIGURE 2: External Web Server Connector Wizard, Step 2

What You Need to Make It Work

The beauty of using servlets with ColdFusion is that it requires few or no special configurations on the part of the ColdFusion server or the servlet server.

To compile the Java code you'll need a JDK, which can be obtained from Sun Microsystems' JavaSoft Web site (http://java.sun.com/j2ee/j2sdkee/), as discussed in Part 1 of this series (***CFDJ***, Volume 3, issue 1). In addition, you'll need the Servlet SDK (a complete version can be downloaded from the JavaSoft Web site, www.javasoft.com/products/servlet/index.html), or you can use JAR files that accompany either Allaire's JRun or the Apache Group's Tomcat products. It's important for the purposes of this article that the servlet library is the 2.2 specification or later. While it's possible to do almost all of the examples in this article on a prior version, the JRun integration is more demanding.

The server you're using – Allaire's JRun or the Apache Group's Tomcat – will determine the default location to install the servlet examples from this article. For JRun, where the base installation directory is <JRUN>, it's <JRUN>\servlets or <JRUN>\servers\default\default-app\WEB-INF\ classes. For Tomcat, where the base installation directory is <TOMCAT>, it's <TOMCAT>\webapps\ROOT\WEB-INF\classes.



Connecting ColdFusion Server and JRun

While JRun hosts its own Web server and allows for direct connects on a configured port, it also allows for connections through other Web servers, which act as proxies for servlet requests. This intermediary technique allows JRun to better cooperate with other servers. Configuring for interaction between the ColdFusion Server and JRun to enable the use of the <CFSERVLET> tag requires a little tinkering.

It takes four quick steps to configure the external Web server connector, and you're up and running. This process begins by launching the Connection Wizard from the toolbar of the JRun Management Console.

The first step in configuring an external Web server connector is the selection of "Server." This panel prompts for the selection of a JRun Server and an external Web Server (see Figure 1). The JRun Admin Server and JRun Default Server are your stock options for JRun Server, and it's strongly advised that you configure for the Default Server. The Web server options are Apache, Netscape (Enterprise or FastTrack), Microsoft (IIS or PWS), O'Reilly WebSite Pro, or JRun's own Zeus. I'm connecting the JRun Default Server to IIS 5.0 on Windows and Apache 1.3.x on RedHat.

At this point the Connector Wizard prompts you to shut down your Web server before proceeding.

The second step presents a panel for the selection of the address and port number on which the configured JRun Server listens (see Figure 2). I'm configuring for address 127.0.0.1 on port 88.

The content in the third step is dependent upon the Web server selected in the first step. This panel prompts for the installation location and the type of connector plug-in (for example, Figure 3 for IIS on my Windows installation, and Figure 4 for Apache on my Linux installation).

The final panel is merely a success or failure notice along with a summary of the selected configurations (see Figure 5).

A high-level view of this configuration is available in its own panel (see Figure 6). You can access it by unfolding the Default Server parent node in the navigation tree and selecting the "External Web Server" node. From this panel we can administer all the settings we entered in the Connector Wizard. Return to this panel if you need to change the listening port.

To inspect the settings of JRun's Default Server you can access it by unfolding the Default Server parent node in the navigation tree and selecting the "JRun Web Server" node (see Figure 7). From here we can look up the port number the Web server is listening on. For the purposes of this article, mine is set to 8100.

The Quick and Dirty of a Java Servlet

A Java servlet is a compiled Java class that inherits from the javax.servlet.http.HttpServlet class. The kinds of requests the servlet is to handle (e.g., HTTP GET and POST) will determine which methods the HttpServlet subclass overrides, doGet or doPost. These methods receive two parameters, a request object and a response object. Passed arguments can be drawn from the request object, and an output stream object can be created from the response object.

The code in Listing 1 (Code Listings 1–6 appear on the ***CFDJ*** Web site) shows a servlet written to respond to both GET and POST methods and will output a simple "Hello, World" string. It takes an optional parameter, Name. I've overridden both doGet and doPost to forward to the same class method, since the different means of calling servlets from ColdFusion require flexible routing, as we'll see later.

Connecting to a Servlet

There are a number of ways HTML and CFML can incorporate servlets. Any tag that can connect to a URL is a viable option. The HTML tags for anchors, form actions, and the implementation-specific <SERVLET> tag, and even image sources and the CFML tags <CFLOCATION>, <CFFORM>, <CFHTTP>, and <CFSERVLET> present us with a number of possibilities. The two that we'll discuss are <CFHTTP> and <CFSERVLET>. These tags allow a controlled two-way communication between CFML code and the servlet.

Serving It Up

I'll discuss how to make your ColdFusion application integrate generically with servlet engines, then focus on the specific benefits of using <CFSERVLET> and the JRun server. While the techniques discussed for use with <CFHTTP can be applied to any servlet engine, use of <CFSERVLET> can only be done with JRun.

Where possible, I've tested with both JRun 3.0 SP1 and Apache's Tomcat 3.2 extension under both Windows 2000 Professional and RedHat Linux 6.2. (The details of the configurations are covered in the first article of this series.)

The dizzying array of port numbers passed about in this article can create conflict. In all cases my Web server listens on the default port (80). The rest of my configurations are as follows:

- JRun Default Server on port 8100
- JRun External Connector on port 88
- Tomcat Connector on port 8080

This collection of port numbers allows me to run all but my default Web server at once.

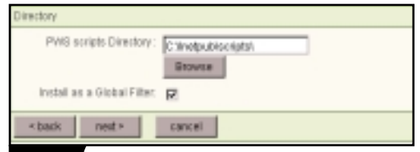


FIGURE 3: External Web Server Connector Wizard, Step 3 (IIS)

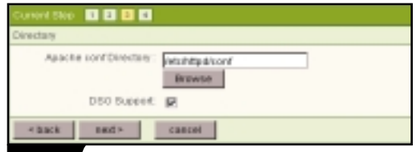


FIGURE 4: External Web Server Connector Wizard, Step 3 (Apache)

FIGURE 5: External Web Server Connector Wizard, Step 4

Web Server to Web Server

The <CFHTTP> tag is powerful in that it's Web-technology agnostic. Through <CFHTTP> we can connect to a servlet without the need (or conversely, the benefit) of the advanced hooks that Allaire provides with JRun and the <CFSERVLET> tag.

Calling a servlet with <CFHTTP> requires a minimum of three attributes: URL, Port, and Method. As might be inferred, the URL attribute specifies the URL to the servlet itself; Port is the port the target server is listening on for the servlet request; and Method is the HTTP GET or POST method.

To call the servlet from Listing 1 without a parameter using JRun:

```
<CFHTTP
URL="http://localhost/servlet/List-
ing1" Port="8100" Method="GET">
```

And with Tomcat:

```
<CFHTTP
URL="http://localhost/servlet/List-
ing1" Port="8080" Method="GET">
```

We can programmatically access the resulting string from the CFML variable, CFHTTP.FileContent.

Passing values to the servlet is equally easy and requires the use of the tag <CFHTTPPARAM>. This sub-

ordinate tag requires three attributes: Name, Value, and Type. The Name attribute specifies the name of the servlet parameter being passed; the Value is the value of the servlet parameter; and Type is the mode that the parameter will be passed. Available modes are: URL, FormField, Cookie, File, and CGI.

Here's an example of this with JRun:

```
<CFHTTP
URL="http://localhost/servlet/List-
ing1" Port="8100" Method="POST">
 <CFHTTPPARAM Name="name" Value="Guy"
Type="FormField">
</CFHTTP>
```

And with Tomcat:

```
<CFHTTP
URL="http://localhost/servlet/List-
ing1" Port="8080" Method="POST">
 <CFHTTPPARAM Name="name" Value="Guy"
Type="FormField">
</CFHTTP>
```

Notice that the Method attribute of <CFHTTP> changed to POST. Usage stipulates that in order to call <CFHTTPPARAM> the Method must be POST. This also means that the servlet's doPost class method will get invoked.

FIGURE 6: JRun Default Server's external Web server configuration panel

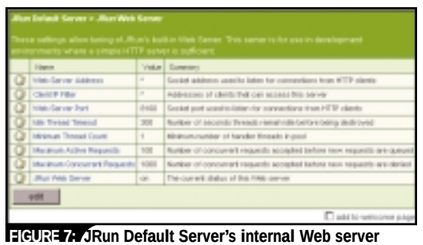


FIGURE 7: JRun Default Server's internal Web server configuration panel

Leveraging JRun

With <CFSERVLET>, a template can not only call a servlet but also get values returned directly into variables. This powerful feature alone allows you to create robust applications. To call a servlet with <CFSERVLET> requires a minimum of four attributes: Code, JRunProxy, Timeout, and WriteOutput. The Code attribute specifies the name of the servlet CLASS file (or its configured alias); the JRunProxy attribute specifies the address and port where the JRun external connector server is listening; Timeout is the number of seconds to wait before reporting a failure; and WriteOutput is a Yes/No value that specifies whether the servlet's output is entered directly into the stream back to the Web server (the default value is Yes). If WriteOutput is set to No, then the servlet output can be retrieved from the CFSERVLET.Output variable.

All servlets called with <CFSERVLET> are called with the HTTP GET method, meaning that its doGet class method will be invoked.

An example of this is:

```
<CFSERVLET Code="Listing1" JRun-
Proxy="127.0.0.1:88" Timeout="300"
WriteOutput="Yes">
</CFSERVLET>
```

Passing parameters to a servlet requires the subordinate tag <CFSERVLETPARAM>, which has a minimum of two attributes: Name and Value. The Name attribute is the name of the passed parameter and Value is its value.

```
<CFSERVLET Code="Listing1" JRun-
Proxy="127.0.0.1:88" Timeout="300"
WriteOutput="Yes">
<CFSERVLETPARAM Name="name"
Value="Guy">
</CFSERVLET>
```

Unlike <CFHTTPPARAM> values, <CFSERVLETPARAM> can be linked to CFML variables, and values can be received as well as sent. This is an important factor in constructing robust applications and nearly as simple as the snippet in Listing 1. The <CFSERVLETPARAM> needs an additional attribute with which to specify the output location. The Variable attribute must identify an existing variable to receive the returning value. Attribute combinations of Name, Value, and Variable are illegal, so you must specify the ingoing value with one <CFSERVLETPARAM> statement and specify the outgoing variable with another.

On the last line of the doGet class method the servlet in Listing 2 shows a call to the setAttribute method of the incoming Request object. This call sets up the return of a value through the JRun server to the calling template.

```
<CFSET str="In">
<CFSERVLET Code="Listing2" JRun-
Proxy="127.0.0.1:88" Timeout="300"
WriteOutput="Yes">
<CFSERVLETPARAM Name="str"
Value="in">
<CFSERVLETPARAM Name="str"
Variable="str">
</CFSERVLET>
<CFOUTPUT>#str#</CFOUTPUT>
```

The <CFSERVLETPARAM> tag also has another important attribute: Type. By default ColdFusion passes strings into a servlet; the Type attribute allows for the preservation of specific data types. The Type attribute can be INT, DOUBLE, BOOL, DATE, or STRING.

Here's an example of sending in a Boolean value:

```
<CFSET str="In">
<CFSET flag="false">
<CFSERVLET Code="Listing3" JRun-
Proxy="127.0.0.1:88" Timeout="300"
WriteOutput="Yes">
<CFSERVLETPARAM Name="str"
Value="in">
<CFSERVLETPARAM Name="str" Vari-
able="str">
<CFSERVLETPARAM Name="flag" Vari-
able="flag" Type="BOOL">
</CFSERVLET>
<CFOUTPUT>#str#</CFOUTPUT>
```

Toggling the value of the "flag" variable between "True" and "False" determines whether there's a modification made to the returning "str" variable. Examining the servlet in Listing 3, you can see that just after displaying the incoming parameter "str", the "flag" attribute is retrieved. After some conversion between the Java Boolean object and Boolean primitive, the code evaluates the True/False condition and determines whether to set the outgoing value.

Another example of passing typed variables can be found in the JRun documentation, *Developing Applications with JRun*, Chapter 42, "Using JRun with ColdFusion."

Wrapping It Up

It's possible to leverage existing servlet resources or even whole applications without too much difficulty. Further, if the servlet server is JRun, it only takes a little tinkering to have strong interaction between ColdFusion templates and servlets. Leveraging existing servlets (many of which are free) can push your ColdFusion application over the edge, with toolkits for generating PDF documents, graphs, images, and more.

Compiled in Listing 4 are all the CFML snippets from this article. Listing 5 is a more complex servlet designed for tracing calls through a servlet and its life cycle, similar to the SnoopServlet that comes with both JRun and Tomcat. You can use this for further study of how servlets work, specifically their interaction with other Web technologies. Listing 6 is a simple modification of Listing 3 to call the tracing servlet from Listing 5.

> "It's possible to leverage existing servlet resources or even whole applications

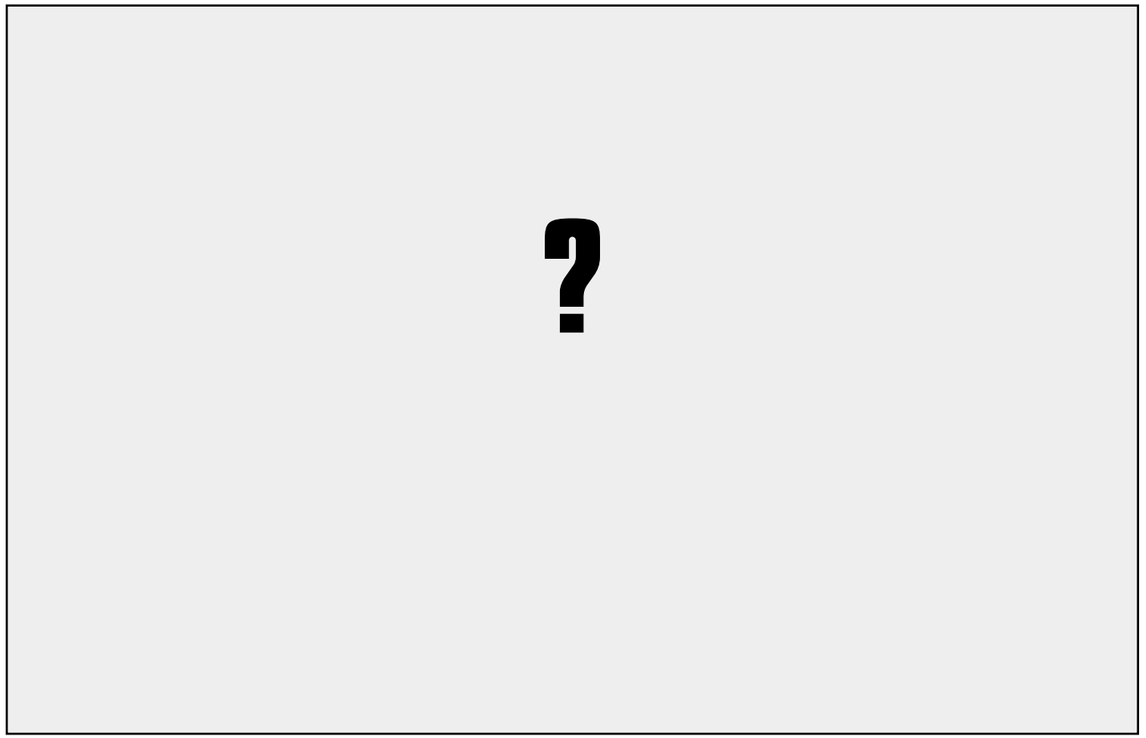
In testing the code for this article I ran into an interesting problem. I originally created a WAR that was to be made available from the ***CFDJ*** Web site. Unfortunately this didn't work very well with the <CFSERVLET> examples. After hunting through documentation and scouring the forums it appears that deployed WARs aren't easily accessible with <CFSERVLET>. According to a post by Bob Love at Allaire's JRun Forum (http://forums.allaire.com/JRunconf/Index.cfm?Message_ID=578270), the only WAR that JRun recognizes for <CFSERVLET> is the default application. This is one of the two default locations that <CFSERVLET> will look for servlets to execute. This fact was only recently reiterated in the <CFSERVLET> documentation found in ColdFusion Studio 4.5.2 (which was released at the time this article was written).

After talking with sales engineers from Allaire I found that it's possible to amend the CLASSPATH setting of JRun's Default Server to include the servlet subdirectory of a deployed WAR. This allows servlets from various WARs to be found by <CFSERVLET>. I didn't find this to be an acceptable solution, however, since the global settings of the application aren't inherited in this way.

ABOUT THE AUTHOR

Guy Rish teaches and mentors ColdFusion and object-oriented analysis and design for Andrews Technology, a consulting firm in San Francisco. He holds instructor certifications from Rational Software and Allaire and teaches ColdFusion for SFSU in the Multimedia Studies Program. Guy is active in both the ColdFusion and OO communities through user groups and was a workshop leader at UML 2000.

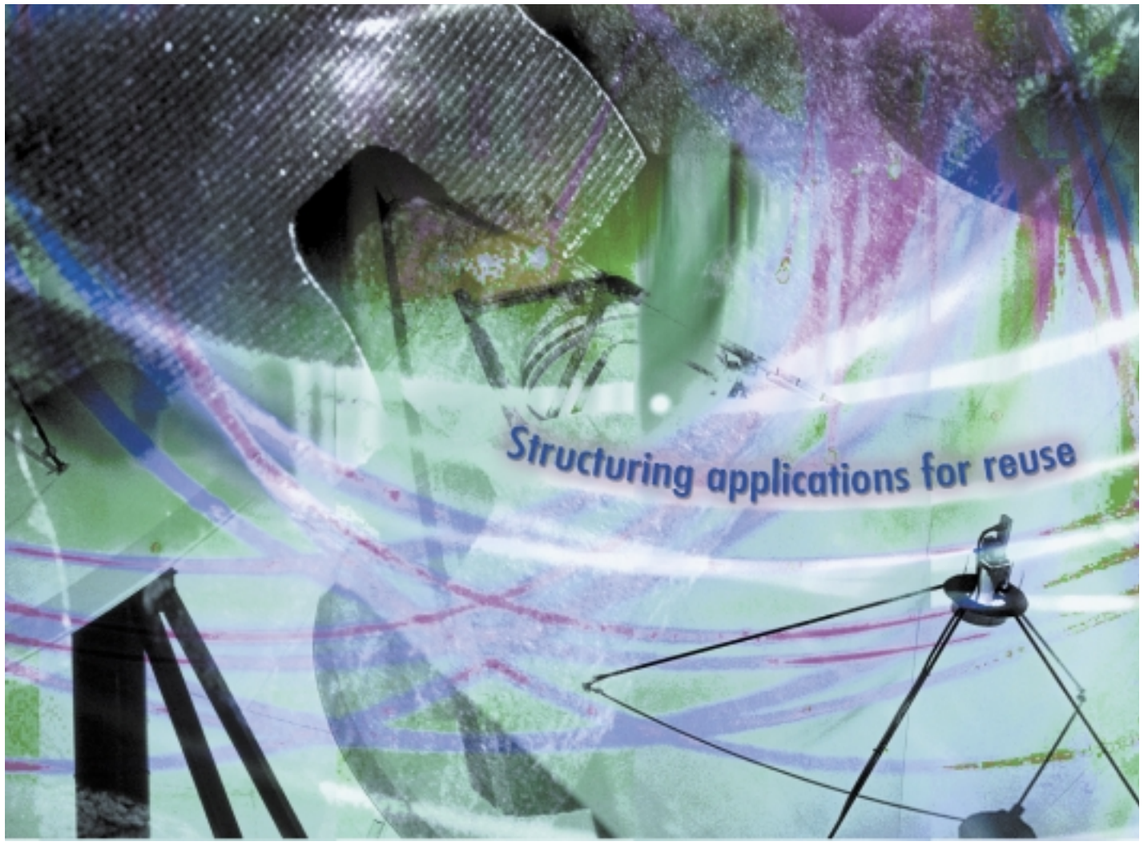
GRISH@CNCDSL.COM



BY RALPH FIOL

M-Commerce:

A Practical Approach to Usability



For the most part, Web applications don't port to wireless applications very well. That is, while Web content and data can be ported to WAP, Web page presentation and navigational elements cannot. Besides the syntactical differences between HTML and WML/HDML, WAP applications are limited by the current technology of wireless devices. Small screen sizes, limited input capabilities, and limited bandwidth make it nearly impossible to port Web content to WAP content directly.

Developers need to rethink how end users will interact with data and how to present it to them. The limitations of WAP become painfully apparent when complicated applications are put under the constraints of the wireless environment. For example, most device displays, such as mobile phones, can show only three or four lines of text (possibly six or seven on newer devices) with limited line length. In addition, the connection speed for the current generation of wireless devices is notoriously slow.

This fact reinforces the need for proper application partitioning. A well-designed application should separate the presentation of data from the access to, and manipulation of, that data. The ultimate goal of proper application partitioning should be reuse. Centralizing logic into code libraries or a middle tier means that we can share (reuse) core system components between our WAP and Web applications. Several tools aid in reuse by moving logic from the application code into the mod-

?

ules or components, such as database-stored procedures, COM objects, and object-based technologies. For ColdFusion developers, methodologies such as Fusebox (www. fusebox.org) and frameworks such as cf-Objects (www.cfobjects.com) make it possible to structure our applications for reuse. cfObjects specifically makes it easy to create class libraries that can be used across multiple applications. In the example presented in this article, the Web online store (store.activetier.com) was constructed using cfObjects. The mobile application (mobile.activetier.com) uses many of the same classes.

Case Study: mobile.activetier.com

In this article we present a case study that walks through the process of taking a full-featured, Web-based e-commerce application and creating a wireless interface. We'll focus on architecting the mobile application to make optimal use of the small screen size, limited input capability, and limited bandwidth of the technology. We'll suggest three design guidelines that will facilitate the identification and development of any mobile application.

The geography of the Web-based application is fairly typical (see http://store.activetier.com). As an HTML-based application, it includes many bells and whistles and advanced features designed to keep the visitor engaged, including:

- ***Browse by category:*** Products are organized by categories, which can have any number of subcategories.
- ***Keyword searching:*** Uses Verity to provide full-text searching of the product database.
- ***Featured products:*** The home page shows three products presented randomly from a pool of featured products.
- ***Shopping cart:*** Shoppers can add to or remove from the cart, select shipping options, and calculate taxes.
- ***Wish lists:*** Visitors can create wish lists or gift registries.
- ***Community section:*** Includes contents, polls, news, and discussion forums.
- ***Content management:*** Portal-like tools for managing news, press releases, and general content.
- ***Product page:*** Shows picture, zoom window, description, SKU combinations, and pricing details.

The Challenges

Given the current limitations of WAP, the challenges of implementing these features for a wireless device are significant. Unless your application is being designed for a specific mobile device, careful thought

FIGURE 1 Home page

has to go into planning what data is essential to the user and how to present it. It's not enough to create a new stylesheet for your application and then design the decks and cards. A complete evaluation of your data and application flow must be done before trying to implement your m-commerce solution. During your evaluation, it will be helpful to keep two things in mind and determine how they'll impact your application.

First, most mobile devices weren't designed as Web browsers. Most mobile phones, for example, were designed to display 10 to 12 characters for the phone number being dialed and one or two lines of text, but not much else. The microbrowsers that mobile phones use are limited to two customizable input buttons and the alphanumeric keypad. On some phones, the task of using the back button requires the user to execute a key combination, so time needs to be devoted to analyzing the navigation of a WAP application. Data input for anything other than numbers is cumbersome and could take the user considerable time, so it should be kept to a minimum. If you have a WAP-enabled phone, try typing the URL for the example application into the phone and you'll understand what we mean.

The second hurdle for your application to overcome is the device cache size. The key flaw in the WAP model is that each phone has a different cache size. When designing cards and decks, keep a close eye on the size of the deck. Deck sizes over 1,200 bytes (yes, bytes) will make your application unusable by some older mobile phones and require the browser to make several requests to the server, which will cost your users not only time waiting for requests to download and display, but also money. In WAP application design sometimes less is more.

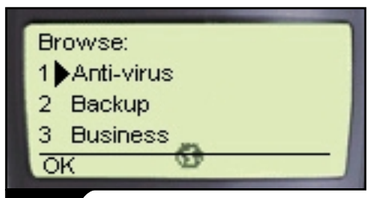


FIGURE 2 Browse

Three Design Guidelines

1. *Wireless applications should be "W+WWW" (wireless + Web).*

Because of the limitations of the technology, standalone wireless applications probably don't make much sense. Instead, developers should provide companion WWW applications and implement only the top 25% of the functionality as wireless. A good example is the commerce application presented here. Focus on the core functionality a mobile user might need – searching, buying, and checking order status. Implement more input-intensive and less frequently needed functions such as profile settings, discussion boards, and customer service functions as Web-only features. Amazon.com does a great job of this. The one-click setting available on their Web site makes it easy to shop their mobile site.

2. *Design your wireless site using a tree structure.*

Unlike the Web, wireless applications have limited navigational aids. A typical mobile phone has only two programmable soft keys. While you can define hyperlinks using anchor tags, they're difficult to use (from a user's perspective), and their implementation varies from device to device. Thus, design your applications to follow a drill-down hierarchical menu that can be accessed using one-click select options.

3. *Minimize data entry*.

Wireless devices have poor user input capabilities. Nothing will frustrate a WAP user more quickly than having to enter large amounts of data using the keypad on a phone. Thus your goal as a developer is to minimize the amount of data entry required by the user. There are two basic approaches to solving this problem. First, you can use SELECT/OPTION tags to create a menu system that allows the user to easily navigate and drill down into your application. This approach follows Design Guideline 2, above. Second, you can prefill, or have the user prefill, data elements offline as suggested in Design Guideline 1. In our m-commerce application this might mean providing a short PIN number that users can enter during checkout to access their profile previously set up using a Web interface.

Walkthrough of Wireless Implementation

The following sections detail the implementation of the m-commerce components of the e-commerce Web site. As suggested in our design guidelines, we'll implement only the core components of the Web application in our wireless site. Since this is an e-commerce site, selling products is top priority. Thus we'll focus our efforts on:

- Searching and browsing
- Getting pricing information and making purchases
- Checking order status

Searching and Browsing

In the Web-based e-commerce site (store.activetier.com) the user can locate products either by searching via keywords or drilling down into categories and subcategories. We'll implement the same concepts in our wireless application but with variations in the implementation details designed to make optimal use of the technology. Table 1 summarizes the differences in the implementations.

Let's look at the implementation of the Browse by Category feature. As shown in Figure 1, the home page of the application allows the user to browse, search, or check status.

When the user selects the Browse option, the system queries the database to get all categories, subcategories, sub-subcategories, and so on. It then builds one deck, which contains a card for each found category (see Figure 2).

This is an important design consideration. The advantage of this method is that it means only one round-trip to the server. The user can then navigate through all product categories "offline" and won't have to wait for the network each time a category is selected. This is a departure for us as developers of Web applications. Normally for an e-commerce application, we pass URL parameters to the server each time a category is selected in order to bring back the next set of subcategories and products. That makes sense on the Web due to the page-centric nature of HTML applications. However, WML allows us to build multiple subpages (cards) within one page (deck). For example, imagine we have a category hierarchy such as this:

```
Anti-virus Software
        Home
        Network
Backup
...
Business
        ...
```

We can efficiently represent this in WML using multiple cards (all within one deck) as shown in Listing 1. Notice that each category that contains subcategories renders an OPTION tag with a relative target. For example, selecting Anti-Virus jumps to a local card named "#C1", whereas selecting Backup (which doesn't have any subcategories) jumps to a new file named list.cfm (which lists all the products within that category).

	Web Based	Wireless
Searching	Uses Verity to search a collection built from the product database. Allows the user to narrow keyword search by optionally selecting a manufacturer or category from a drop-down control.	Also uses Verity to search the same collection, but due to small-screen real estate, doesn't include "advanced" search options.
Browsing	Allows the user to browse products by category. A product category (e.g., Software) may have any number of subcategories (e.g., Word Processing). When a category is selected, the system queries the server and displays all subcategories, plus products under that category.	Also allows the user to browse products by category. However, to avoid multiple trips to the server, the application queries all categories and subcategories once and builds a deck with multiple cards – one for each possible category. Thus, the server is hit only once, but the user has the same drill-down experience. When the user reaches a category that has no subcategories, the products are displayed.

TABLE 1 Differences in Web-based and wireless implementations

To generate the WML cards shown above, we created a rather complicated query that returns all categories listed in the category table. In our application the category table has two import keys. The first, cat_id, is an identity column and is the primary key of the table. The second is parent_cat_id, which is a foreign key that points back into the category table to identify the parent (if any) category for a subcategory. If a category doesn't have a parent (i.e., it's a root category), then the parent_cat_id column will be NULL. The trick here is to select all categories sorted by parent_cat_id (the root categories that contain a NULL parent_cat_id will be listed first). Then when generating our deck, we use the CFOUTPUT GROUP attribute to build our cards, one card for each group break. The only other tricky part is that we want to know the name (cat_title) of a category's parent category for display purposes. For this, we have to join the category table with itself to get its parent (remember to do an outer join!). We also perform a subquery in order to count the total number of subcategories for each parent category. The complete SQL statement is shown in Listing 2. As you can see, this query is getting a bit ugly. Be sure to create indexes on your key fields and consider implementing this as a stored procedure. Using CACHEDWITHIN wouldn't hurt either.

The next step is to generate the WML from this query. Remember, we've sorted the result set by parent category so that all same-level categories are grouped together. Thus we can easily use CFOUTPUT's GROUP attribute to build one card for each group of options. The code is shown in Listing 3.

We now have a complete drill-down application contained within one deck. You can try it out by pointing your mobile device or simulator to http://mobile.activetier.com. If you're using a simulator such as the one provided in the UP SDK (www.phone.com/products/upsdk.html), try using the View Source option to see the code generated by the code in Listing 3. One final word of caution regarding this implementation: if you have many categories and subcategories, you run the risk of exceeding the maximum deck size. If that's the case, you can experiment with different options, such as:

- Add a new column to your category table to store an abbreviated title or "short title" for WAP browsing. This may save you some precious bytes.
- Retrieve only two levels of categories at a time.
- Retrieve all top-level categories, but only the most popular subcategories. Allow the user to drill down to less popular categories by making an additional trip to the server only as needed.

Searching

We've used Verity to implement full-text searching of our product database. And we've used the same collection previously built by the Web e-commerce site within our mobile site. The search form in Figure 3 simply has one input field and a "Search" button. The WML code in Listing 4 creates an input control and uses GO/POSTFIELD to submit the keyword(s) to the server.

Next we use Verity to search the collection and return matches (see Listing 5). Although not shown here, we take the results from Verity and extract just the primary key data using valueList(q_results.custom1). We then use just the primary keys to query the product database to get details, such as price and SKU data. Finally, we build a select list for each product that shows just the product title. The user can select any product to go to the item.cfm page to view product details, such as available SKUs and pricing (see Figure 4).

A final comment on searching: you may want to consider adding additional search methods. For example, in our implementation, we allow the user to search only by

keyword. You may want to allow the user to search by SKU number, inventory number, or part number. A growing practice in e-mail marketing is to include not just a URL to a specific product for Web surfers, but also an item number or small three-digit code for mobile surfers. That way, wireless surfers can quickly navigate to your featured item by entering a short numeric code.

FIGURE 3: Search form

Getting Pricing Information and Making Purchases

Of course, after implementing searching and browsing, we need to provide the user with a way to view a product, get pricing information, and make purchases. Figure 4 shows the first screen a user sees when a product is selected. In this example the user has selected Adobe PhotoShop 6.0 from a search screen or some other product listing. Here, we show all SKUs available for that product as a SELECT list. In our catalog we have two versions of Adobe PhotoShop – one SKU for the Mac and one for the Windows platform.

In our data model we have the concept of a product family (e.g., Adobe PhotoShop 6.0) and individual product SKUs (e.g., PhotoShop for Windows 6.0). A product family has one or more product SKUs, and a product SKU belongs to one product family. All pricing is stored at the SKU level, as shown in the display in Figure 4. To build this page, we've once again made use of the ability to create multiple cards within one deck. In Listing 6 we created one card that displays the list of SKUs for the selected product family and another card to allow the user to enter a quantity to be added to his or her shopping cart. And we're making use of WML variables to pass the selected SKU ID (pro_id) to the "Add To Cart" card. You'll note its use in the first POSTFIELD shown in the second card.

Two things to note in this listing. First, I use the ColdFusion function "dollarFormat" to display the price. Recall that this function formats a number with two decimal points and inserts a dollar symbol. Since dollar symbols are used in WML to indicate variables, I've added a leading hard-coded dollar symbol so the result generated after ColdFusion has processed this file is a double dollar symbol. The WML compiler will then see the back-to-back symbols and correctly treat the first as an escape character for the second.

The second thing to notice in Listing 6 is that I've added the optional attribute FORMAT to the INPUT tag. Specifying a format mask of "N*" puts the phone's keypad into number-only mode, so the user doesn't have to press the keys repeatedly to cycle through alpha characters to reach the numbers. This also ensures that the user is entering numbers where only numbers belong.

After the shopper has entered a desired quantity, the value is posted to the AddToCart.cfm page, where the product ID and quantity are added to the shopping cart. Then the user is shown the contents of the cart, and he or she can continue shopping or proceed to the checkout process.

The Checkout Process

The checkout process in a Web-based e-commerce site typically involves signing in, entering a billing address, entering a shipping address, reviewing your order, and entering payment information. A well-designed site allows the shopper to set up a profile so most of this information can be stored and prefilled on subsequent visits. Nevertheless, checkout is normally a complicated and fairly interactive process.

How Would Bezos Do That?

When deciding how to best implement this process for WAP, I took a quick survey of the m-landscape to see how the titans of e-commerce approached the problem. To my surprise, I found several different implementations, each with some pretty cool ideas. One of my favorites turned out to be the simplest to implement and makes good use of the media. At Buy.com's mobile portal users who haven't already registered can connect with a live operator to set up an account. That is, new users who don't want to sprain a thumb entering billing, shipping, and credit card information can simply click the "Call" soft key to connect to a live operator who will do the work for them. To implement this in your own site, simply include a DO tag as follows:

```
<do type="accept" label="Call">
  <go href="wtai://wp/mc;3054448324"/>
</do>
```

This assumes that the WAP device is a phone. It calls a system function that commands the phone to dial the number listed.

Amazon's site allowed me complete control over my online account using my wireless device. Because I already had an existing account with Amazon, I was able to log in once, and the site remembered me from that point on, showed me the same recommendations I would have seen on the WWW site, and – best of all – allowed me to shop using my one-click settings.

FIGURE 4: View product, select SKU

CDNOW took yet another approach. The company's WAP portal doesn't provide any type of phone-based registration. Users must go to the Web site to register an account and turn it on for one-click shopping.

Summary

Applying wireless Internet technology requires some fairly complex design challenges. As developers, we'll have to rethink the way we design and organize our applications, with specific focus on tight code, navigation, and utility. Rather than drowning the user in information the way the Web does, developers and content providers have to be sensitive to what the user wants and how to get to it.

Following the three design guidelines – W+WWW, create a tree structure, and minimize data entry – developers can take a structured approach to building an m-commerce application. During each step in the construction special consideration should be given to minimizing the deck size, the amount of network traffic (trips to the server), and the amount of data entry required by the user.

Acknowledgment

Many thanks to Alex Leon for his contributions to this article. Alex is the senior programmer for New Media Studio, a South Florida–based Web development firm specializing in high-end design and e-commerce solutions. He can be reached at aleon@newmediastudio.com.

About the Author

Ralph Fiol is CTO for ActiveTier Technologies, Inc., an e-services company that helps its clients identify and create sustainable, competitive advantage through the application of technology. He is a certified ColdFusion instructor and the author of the Open-Source framework "cfObjects, an object-oriented framework for ColdFusion development."

ralph@activetier.com

?

Listing 1: Multiple cards for categories

```
<card id="c0" title="Browse">
<p>Browse:
<select title="category">
<option onpick="#c1">Anti-virus</option>
   <option onpick="list.cfm?c=2">Backup</option>
<option onpick="#c2">Business</option>
  </select>
 </p>
</card>

<card id="c1" title="Browse">
<p>Anti-virus:
<select title="category">
<option onpick="list.cfm?c=11">Home</option>
<option onpick="list.cfm?c=10">Network</option>
  </select>
 </p>
</card>

<card id="c2" title="Browse">
<p>Business:
<select title="category">
<option onpick="#c3">Suites</option>
<option onpick="#c4">Financial</option>
  </select>
 </p>
</card>
```

Listing 2: Get all categories

```
select  C1.*,
   C2.cat_title as parent_title,
   (select        count(*)
    from          category C3
    where         C3.parent_cat_id = C1.cat_id and
         C3.cat_x_active = 1
   ) as total_subcategories
from    category C1,
   category C2
where             C1.cat_x_active = 1 and
   C1.parent_cat_id *= C2.cat_id
order by
   C1.parent_cat_id,
   C1.cat_title
```

Listing 3: Build one card for each category

```
<cfoutput query="qCategory" group="parent_cat_id">
 <card id="c#val( qCategory.parent_cat_id )#" title="Browse">
  <p>
  <cfif len(trim(qCategory.parent_title)) eq 0>
   Browse:
  <cfelse>
   #trim(qCategory.parent_title)#:
  </cfif>
   <select title="category">
   <cfoutput>
    <cfif qCategory.total_subcategories EQ 0>
     <option
onpick="list.cfm?c=#qCategory.cat_id#">#trim(qCategory.cat_ti
tle)#</option>
    <cfelse>
     <option
onpick="##c#val(qCategory.cat_id)#">#trim(qCategory.cat_title
)#</option>
    </cfif>
   </cfoutput>
   </select>
  </p>
 </card>
</cfoutput>
```

Listing 4: Search form

```
<card id="search" title="Search">

 <!---/// Reset keywords  ///--->
 <onevent type="onenterforward">
  <refresh>
   <setvar name="keywords" value=""/>
  </refresh>
 </onevent>

 <!---/// Ask the user for input ///--->
 <p>Keywords:
  <input type="text" name="keywords" value=""/>
 </p>

 <!---/// Post data to search page ///--->
 <do type="accept" label="Search">
  <go method="get" href="search_p.cfm">
   <postfield name="keywords" value="$(keywords)"/>
  </go>
 </do>

 <!---/// Back Button ///--->
 <do type="prev">
  <go href="index.cfm"/>
 </do>
</card>
```

Listing 5: Search procedure

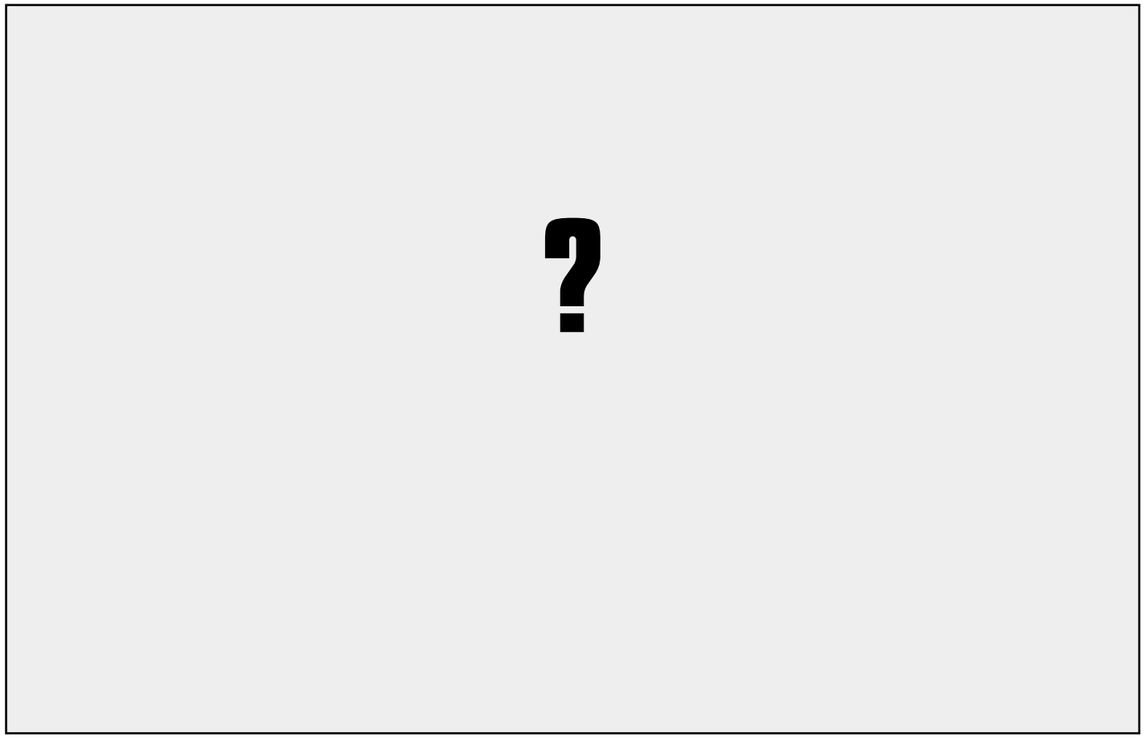
```
<cfsearch collection="store_products"
  type="SIMPLE"
  criteria="#lCase(trim(keywords))#"
  name="q_results"
  maxrows="50">

<!---/// Next, query database to remove duplicates
 and get more details.
///--->

   ...

<!---/// Next, display results ///--->

<card id="results" title="results">

 <cfoutput>
 <p>Found #numberFormat(q_products.recordCount)#
 </cfoutput>

 <select title="View">
 <cfoutput query="q_product">
  <option onpick="item.cfm?pf_id=#q_product.id#">
   #trim(q_product.title)#
  </option>
 </cfoutput>
 </select>
 </p>

 <!---/// Back Button ///--->
 <do type="prev">
  <go href="search.cfm"/>
 </do>

</card>
```

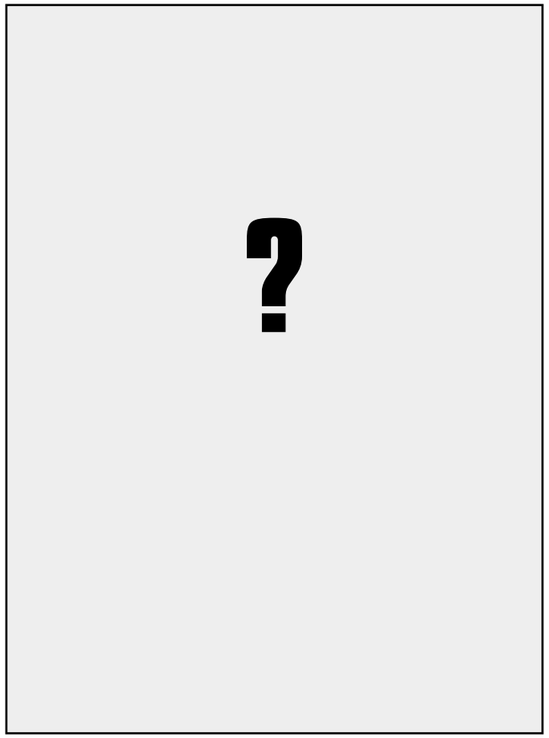
Listing 6: Product information

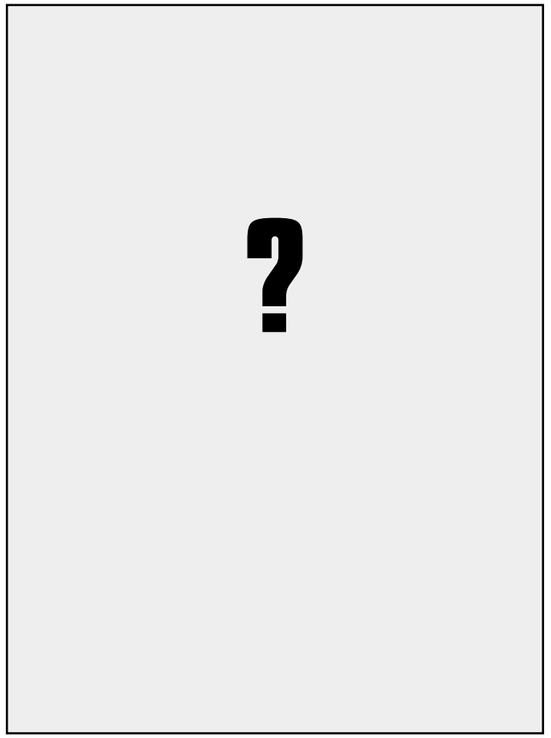
```
<template>
 <do type="prev" label="Back">
  <prev/>
 </do>
</template>

 <!---/// Main Menu ///--->
 <card id="item" title="Item">
  <cfoutput>
  <p>#trim(q_family.pf_title)#
   <select name="pro_id">
   <cfloop query="q_sku">
    <option onpick="##addtocart"
value="#q_sku.pro_id#">#trim(q_sku.pro_title)# $#Dollar-
Format(q_sku.pro_price_retail)#</option>
   </cfloop>
   </select>
  </p>
  </cfoutput>
 </card>

 <card id="addtocart" title="Buy">
  <!---/// Get Quantity ///--->
  <p>
   Quantity:
   <input name="pro_qty" value="1" format="N*"/>
  </p>

  <!---/// Button ///--->
  <do type="accept" label="Buy">
   <go method="get" href="addtocart.cfm">
    <postfield name="pro_id" value="$pro_id"/>
    <postfield name="pro_qty" value="$pro_qty"/>
   </go>
  </do>

 </card>
```

CODE LISTING

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







Ask the **Training Staff**

A source for your CF-related questions

BY
BRUCE VAN HORN

My thanks to those of you who've read this column and let me know that it's actually helped. Even more thanks to those who've sent in questions – keep them coming! Here are some of the most recent ones.

Q: *I need to create a server-wide variable, not a session variable. It cannot be user specific. Can this be done without using a table?*

A: Most certainly! That's what the server variable scope is used for. Most programmers just starting out with CF quickly discover how to use session and application variables, but the server variable is often overlooked. All you need to do is set a variable with the server prefix (i.e., <CFSET Server. YourVar = "Some Value">). Once the variable is set, it's stored in the server's memory and is available to all CF pages running on that server.

Q: *I've installed CF Server (professional version) on my Win98 machine running MS Personal Web Server. When I try to browse any of my CF pages, the actual CF code shows up in the browser, but none of it actually executes. What have I missed?*

A: This is a common problem when installing CF with PWS. The problem is CF doesn't have permission to execute the pages through PWS. The good news is it's easy to fix. All you need to do is launch the PWS administrator and assign execute rights for your wwwroot subdirectory. You can do this by editing the directory properties.

Q: *What does the SETDOMAINCOOKIES attribute of the <CFAPPLICATION> tag do?*

A: You really need to worry about this attribute only if you're running CF in a clustered server environment. It's used in the same sense as the DOMAIN attribute of the <CFCOOKIE> tag. Basically, cookies are server-specific, which means that a cookie will be sent back only to the server that sent it. For example, if your server (shop.yourdomain.com) writes a cookie, the user's browser will return that cookie only to the server that created it (shop.yourdomain.com). If you need cookies to be returned to any server in your domain (???.yourdomain.com), you need to set "Domain-level" cookies.

The SETDOMAINCOOKIES="Yes" attribute of the <CFAPPLICATION> tag makes the two cookies (CFID and CFTOKEN) that are used for session and client variable management domain cookies to allow for session and client variable management across a cluster of servers.

Q: *I need to pass a product ID along the URL, but I don't want the user to know what the value of the ID is because I don't want them to be able to modify the URL value and view other products. Is there a way to keep people from changing a URL variable value or a way to hide it from them?*

A: The answer is both yes and no. No, you can't keep someone from modifying a URL string, but you can write your code so he or she doesn't really know what's being passed along the URL. There are many different ways to do this but perhaps the easiest is to encrypt your URL string so the user doesn't know what's being passed. It's a two-step process. First you need to encrypt the URL variables, then you need to decrypt them so your code on the other side understands what was passed to it. Use CF's Encrypt() and Decrypt() functions.

Listing 1 shows the code for List.cfm, which runs a query and displays product names to the user. Each product name has a link to Details.cfm (see Listing 2) and passes the product ID (in an encrypted format) along the URL. Notice the use of the URLEncodedFormat() function wrapped around the Decrypt() function. This is required because the characters generated by the Encrypt() function may not be valid URL characters. Listing 2 shows how to decrypt the value and use it in a query. The most important part to encrypting and decrypting is to use the same "key" value for both functions.

Please send your questions about ColdFusion (CFML, CF Server, or CF Studio) to AskCFDJ@sys-con.com.

BRUCE@NETSITEDYNAMICS.COM

ABOUT THE AUTHOR
Bruce Van Horn, a regular contributor to **CFDJ,** *is president of Netsite Dynamics, LLC (an Allaire Alliance Partner), and an Allaire certified instructor.*

Q&A *continued from previous page*

Listing 1

```
List.cfm

<CFQUERY NAME="qGetBeans" DATASOURCE="FastTrack_Solution">
 SELECT * FROM Beans
 ORDER BY  Bean_ID
</CFQUERY>

<CFOUTPUT QUERY="qGetBeans">
<A
HREF="details.cfm?ID=#URLEncodedFormat(Encrypt(qGetBeans.Bean
_ID,"key"))#">
#qGetBeans.Bean_Name#</A><BR>
</CFOUTPUT>
```

Listing 2

```
Details.cfm

<CFQUERY NAME="qGetBeans" DATASOURCE="FastTrack_Solution">
 SELECT * FROM Beans
 WHERE Bean_ID = #Decrypt(URL.ID,"key")#
</CFQUERY>
```

CODE LISTING

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

?

Interview...with Kevin Newling
VP of AbleCommerce

INTERVIEWED BY **ROBERT DIAMOND**

CFDJ: Tell us about AbleCommerce and Able Solutions. What are you all about?

Newling: AbleCommerce was founded in 1994. We came across ColdFusion toward the end of that year. We'd originally developed an e-commerce application in C. When we saw how much opportunity there was to use the features of ColdFusion to do e-business, we developed our first version of AbleCommerce in ColdFusion. And that's when we started promoting it publicly.

CFDJ: How has AbleCommerce grown in terms of size and revenue since 1994?

Newling: At its inception, AbleCommerce was an entrepreneurial vision. We contracted with small local companies seeking to enter the uncharted waters of e-commerce. From those inauspicious beginnings, AbleCommerce has evolved into a company that employs 35 people in the U.S. and 15 locations in other countries. Revenue has more than doubled each year for the last several years.

Further, unlike many of the dot-coms that have emerged over the last several years, AbleCommerce has shown a solid profit and growth every year. With the release of two new products toward the end of 2000, we're forecasting phenomenal growth this year.

CFDJ: Where are you today?

Newling: Well, we gained quite a few customers, including Nike, Casio, Compaq, and Hard Rock Hotel, as well as numerous smaller companies seeking to do business online. We've also launched our AuctionBuilder application, which we're proud to say now has a 7% share of the online auction market. Our newest e-commerce solution release is AbleCommerce CommerceBuilder 3.0.

CFDJ: How have companies such as Nike, Compaq, and others with pre-existing Web sites integrated AuctionBuilder with their current offerings?

Newling: Numerous companies of various types have integrated AuctionBuilder into their sites. Some centered on the auction model, such as Music-HotBid, while others are e-commerce sites, such as Jangle.com, that have added auctions to enhance their online community and to act as a clearinghouse for unsold inventory. AuctionBuilder has also been integrated by real-world auction companies such as Felix-Holland in Texas (netauction.com) to add an online component, and there's a real estate company that auctions houses online. One of the most exciting applications of AuctionBuilder has been in the B2B market where companies that once served as the deal facilitator have evolved into entire online exchanges. One such company services the forest industry, another handles telecom hardware and materials, and one has even been created as an exchange medium for sporting event sponsorships and recruiting athlete spokespersons.

CFDJ: What's new in CommerceBuilder 3.0?

Newling: Quite a bit really. It enables companies to be real ASPs or CSPs (Commerce Service Providers). To service this market, we have integrated a new feature called QuickStore. This allows service providers to compete with Yahoo stores and other online store rental services. Users can set up an AbleCommerce Store using the QuickStore feature and rent out stores without ever needing to talk to the actual merchants that come in. Merchants using our 2.9 product would have to know some HTML to build their store. We now have a built-in WYSIWYG (What You See is What You Get) editor, so you don't even need to know HTML. Other major features include a transition to international metrics including multicurrency display, a powerful product kitting tool for bundling component products, and an advanced tax calculation system.

CFDJ: Can you tell us about some component products that have been created?

Newling: Several component products are currently available for integration with AbleCommerce, including resources to enable wireless access as well as tools that increase the core functionality of AbleCommerce. Foresite-design, one of our premier partners, has added capabilities to enable gift certificates and coupons, the creation of affiliate programs, and a global discount system.

We recently added two partners that offer solutions that tie directly into our products. Global CommerceZone has created a plugin that will automatically calculate a landed cost for international sales. Also, through a partnership with iCommunicate, we can offer an ASP-based online customer service system. All these solutions have been specifically tailored to integrate with the opportunities AbleCommerce software products open in the online commerce arena.

CFDJ: Sounds like you're in the running to win another Readers' Choice Award. What's the cost?

Newling: The cost is $2,495 for the source code version, and $995 for the encrypted version. AbleCommerce 3.0 is also available bundled together with AuctionBuilder 1.0 in our suite version for $8,995.

CFDJ: We've heard a lot about ColdFusion 5 and about Neo. Where do you see AbleCommerce going in the future?

Newling: We've been talking with Allaire about the future and how we can take advantage of where ColdFusion is going with our products. We're doing a substantial push internationally. Version 3.0 of AbleCommerce has numerous international features, including a multitier tax structure, the integration of international formats, and multicurrency display to appeal to the international markets. AbleCommerce has 14 international offices: UK, Japan, Greece, Turkey, Malaysia, United Arab Emirates, South Africa, Chile, The Benelux, Hungary, Norway, Spain, Singapore, and Mexico – all local offices serving local markets. Quite a few of these are seed markets, where we're investing in building an e-commerce infrastructure because we know they'll grow. We know that they're maybe two, three, or as many as five years behind us, but we anticipate that they're going to catch up fast and build e-commerce much faster than we did here. So in addition to some phenomenal developer- targeted products carrying the AbleCommerce CommerceBuilder and AuctionBuilder product lines forward, we will continue to strengthen the international offering and build on the commerce service provider market.

CFDJ: What plans are there to WAP or wirelessly enable future versions?

Newling: One of our partners, The CyberGroup, has a plugin that enables full wireless interaction with our software from the administrator side. Through this resource, AbleCommerce merchants can check the status of orders and inventory, while customers can order via their wireless connection. Similar features are available for AuctionBuilder, enabling bidders to receive notification if they are outbid or if they win an auction. Future releases by AbleCommerce will integrate this technology, either as a plugin or directly within the software.

CFDJ: We're definitely looking forward to that. And if people want more information?

Newling: Visit our newly redesigned Web site at www.ablecommerce.com.

robert@sys-con.com

February **2001**

What's Online This Month...

www.coldfusionjournal.com

www.coldfusionjournal.com is your source for industry events and happenings. Check in every day for up-to-the-minute news events and developments, and be the first to know what's going on in the industry.

Digital Edition

Don't have your print edition on hand? Can't wait for the next issue to arrive in the mail? Our digital edition is just what you need. As long as you have your computer with you, you can read ***ColdFusion Developer's Journal*** anytime, anywhere. Looking to research a specific topic? Search our archives – we've got every article that's been published since our premier issue!

ColdFusion Jobs

ColdFusion Developer's Journal is pleased to offer our employment portal for IT professionals. This site gives you direct access to the best companies in the nation. If you're an information technology professional and are curious about the job market, demand privacy, and don't want to waste time, you've found the right site!

Check out our new career center and post your resume, making yourself visible to thousands of HR managers who want to help you grow your career at their companies! Track new jobs from the hottest dot-com and Fortune 500 firms in the country.

SYS-CON Radio Broadcasts from the Allaire Conference

Listen to ***ColdFusion Developer's Journal*** interviews from the Allaire Developer Conference in Washington, D.C., November 4–6. Tune in as host Robert Diamond talks with the movers and shakers of the ColdFusion industry's leading companies, from Fig Leaf's Steve Drucker to CFX Hosting's Mark Click.

SYS-CON Radio interviews Dave Gallerizzo (center) and Steve Drucker (r) from Fig Leaf.

Ben Forta's ColdFusion Tip of the Day

Click here for ColdFusion tips, links, tags, and resources from Allaire's CF evangelist. A new tip every day!

CFDJList

Join our new ColdFusion mailing list community. You and other IT professionals, industry gurus, and ***ColdFusion Developer's Journal*** writers can engage in CF discussions, ask technical questions, and more. Voice your opinions and assessments on topical issues, or hear what others have to say. Monitor the pulse of the ColdFusion industry!

WDDX with JavaScript

Using WDDX to transfer complex data types between the server and client (and back)

BY CHRISTIAN SCHNEIDER

This article demonstrates how to use Allaire's XML serialization technique, Web Distributed Data eXchange (WDDX [described at www.wddx.org]), to transfer CFML data structures from the server to the client, and into JavaScript and back. Generally WDDX's approach is to interchange data of all types (including complex data types such as nested structures, n-dimensional arrays, record sets, or even binary data) between different programming platforms.

Overview of WDDX

WDDX is simply a way of describing data in a text-based form. Technically it's an XML DTD (Document Type Definition) of XML tags that describes simple and complex data types, such as the contents of a nested structure. Many major Web programming platforms are adopting WDDX; it's currently available in CFML, Java, ASP, PHP, and Perl for the server-sided, platform-independent exchange of data. In addition to transferring data between different server platforms, WDDX is also available in client-sided programming environments such as JavaScript. This article focuses on the latter, but first I'd like to continue with the WDDX overview.

You can use WDDX to serialize some ColdFusion data into a WDDX package:

```
<cfwddx action="CFML2WDDX" input=#myComplexArray#
output="myWddxPackage">
```

The result: a WDDX package that represents the content of #myComplexArray# as an XML string is stored in #myWddxPackage#. It's possible to deliver this WDDX package over the Web using text-based protocols, such as HTTP or SMTP, since the WDDX string is XML and therefore plain vanilla text. This is even true when serializing binary data, which automatically escapes and unescapes to a BinHex64 text representation.

This mechanism allows us to interchange (syndicate) data between different Web applications (even when they're written in different languages). Basically the CFML developer doesn't need to think about the XML stuff, since serialization and deserialization are encapsulated into the <CFWDDX> tag (or the WDDX–SDK on other language platforms). Deserialization from a WDDX package (obtained from a <CFPOP> or <CFHTTP> access, for example) back into a CFML data type is as easy as this:

```
<cfwddx action="WDDX2CFML" input=#myWddxPackage# output="myNewData">
```

The WDDX package that's held in #myWddxPackage# is now deserialized into the CFML variable #myNewData#.

Transferring WDDX Between Server and Client

To provide you with a nice example of how to use WDDX on the client with JavaScript and how to transfer data back to the server, I decided to implement an offline browser for ini files. ini files are simply text files that contain sections and entry-value pairs of configuration settings for some applications. To learn more about the ini style see the CF Studio Help on ColdFusion's SetProfileString() and GetProfileString() functions. I also describe how to grab such a file into WDDX and transfer it as an array of structures to the client where it can be viewed, edited, and extended offline, as well as how to commit these changes back to the server in one step for the whole data set. This example enables us to do offline browsing and editing of ini files residing on the server.

Note that this functionality can also be implemented on a stateful online connection with a step-based editing process involving several action pages (such as loading all the sections, the entries of a specific section, and its value, and writing each one back in its own step). However, the goal of this article is to introduce client-sided usage of WDDX using the ini file as an example of structured data that can be transferred at once to the client and back.

Let's look at what our example does. After entering a server-sided path in an ini file (you can also use the demo ini file provided with this example), you'll see two list boxes and an input field (see Figure 1). In the first list box choose a section within that ini file; in the second, a specific entry within that section while the text field shows the value of the selected entry. Also, you can change the values using the button "Update Value" or add entries or even whole new sections using the "New Entry" and "New Section" buttons (see Figure 2). This is all done on the client using JavaScript. There's no backprocessing to the server, not even for adding new sections and entries. Selecting the "Store data back into ini file on server" button immediately sends all data, including the changes and additions, to the server. WDDX serializes the data (see Figure 3), which results in an update of the ini file.

Theoretically you could connect to your server, open an ini file within the offline browser, close the online connection (go offline), change and/or add data to the ini file browser, and finally go online again and commit all changes at once to your server. This is the concept of a simple offline data browser. Note that we're not dealing with concurrent updates, deletes, commits, and rollback statements here, since the goal of this article is to show you how to use WDDX to transfer complex data types between the server and client (and back), not how to implement a perfect offline data browser, which is clearly out of this article's scope. For a nice offline data browser with single-record updating and other interesting features, I recommend Ben Forta's (www.forta.com) book *Advanced ColdFusion Application Development* (Que).

Looking at the Code Details

So far I've explained the concept of using WDDX on the client, but to make this article worth reading it's time to introduce and explain the code behind it. First I'd like to demonstrate how to transfer a CFML array to JavaScript with only a few lines of code:

```
<!--- Include WDDX-SDK for JavaScript --->
<SCRIPT src="/CFIDE/scripts/wddx.js"
language="JavaScript"></script>
```

Before using all the WDDX objects (for example, the WddxRecordSet or WddxSerializer object) within JavaScript, you have to include the above line to import the WDDX SDK for JavaScript in your page. If you use "/" CF mapping, you don't have to worry about which path ColdFusion used to place the wddx.js file. After the WDDX–JavaScript SDK is imported to your page, you can transfer any CFML variables directly into JavaScript. Theoretically this is done with two invocations of the <CFWDDX> tag, as you first have to serialize the CFML data with <CFWDDX action="CFML2WDDX" ...> to WDDX, and then deserialize it with <CFWDDX action="WDDX2JS" ...> into a client-sided JavaScript variable. However, as Allaire's intention is to minimize any coding overhead and to make things as nice and easy

as possible, they created a shortcut to this process that resulted in only a single invocation of <CFWDDX action="CFML2JS" ...>

```
<SCRIPT LANGUAGE="JavaScript"
TYPE="text/javascript">
<!--
 // Make CFML-2-WDDX-2-JS
 <CFWDDX action="CFML2JS" input=#myCfArray#
toplevelvariable="myJsArray">

 // Output length of this array:
 alert(myJsArray.length);
//-->
</SCRIPT>
```

These few lines transfer a complex CFML array (held in #myCfArray#) into a JavaScript array (stored in myJsArray) for client-side usage. To test the code, the length of the JavaScript array is shown in an alert popup box. You can now access all the array elements within JavaScript using myJsArray[index]. But be aware that in JavaScript array elements start at index 0, whereas in CFML they start at index 1. It's also possible to transfer other complex data types, such as record sets or structures, to the client that way. As we're using an array of structures to hold the ini file's sections, note that when accessing structures within JavaScript (coming through WDDX) all keys have to be written in lowercase:

```
<SCRIPT LANGUAGE="JavaScript"
TYPE="text/javascript">
<!--
alert(tmpSection["nameofkey"]);
//-->
</SCRIPT>
```

After this basic WDDX–JavaScript introduction, let's look at the source files for this article. (They can be found on the ***CFDJ*** Web site, www.ColdFusionJournal.com.) Although the example consists of four files (index.cfm, top.cfm, main.cfm, and update.cfm), I'll explain only two of them (main.cfm and update.cfm) since the other two open up only the frame set (Listing 1: index.cfm) and the filename entering form (Listing 2: top.cfm).

The file main.cfm (see Listing 3) is invoked after entering the path and filename of the ini file, which is submitted as FORM.filename. After checking if the file really exists on the server, its content is read completely and transferred into an array of structures through some string-parsing code. This code only reads the ini file structure into a data type that represents the sections and entries of the ini file and has nothing to do with the actual topic of this article. The code is necessary to dynamically read the ini file content and can also be encapsulated into a custom tag, so you don't have to

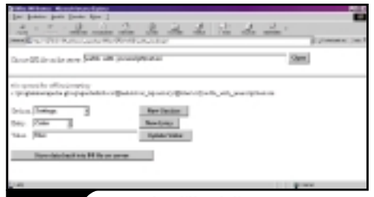


FIGURE 1: Our sample offline inibrowser

worry about it when you're working with WDDX and JavaScript. I won't discuss this parsing code here but will continue the in-depth analysis of the actual WDDX and JavaScript code in main.cfm.

After the ini file has been parsed, its content is available as an array of structures that holds each section, its name, a nested array of structures for each entry, and its corresponding value within the section. This is all stored in the CFML #Profiles# variable, which is transferred through WDDX to the JavaScript ProfilesArray variable. Next you'll see eight JavaScript functions that handle the data on the client side, including the dynamic representation logic and the adding and/or changing of the values, entries, and sections, which I'll discuss in detail later.

In the HTML code there's a <form> pointing to the file update.cfm (which handles the updating of the ini file and is also discussed later). This form has two hidden form fields: one holds the filename and path to the ini file so the update.cfm template knows what file to update, and the other, WddxData, is initially kept empty. *Note:* The hidden form field holding the filename is insecure as it allows evil users to specify other files to overwrite besides the ini file. However, as this example focuses on the WDDX part, I left out the part about security. For the curious, I'd implement security using a SESSION-scoped (and therefore server-sided) storage of the filename, or while still using a hidden form-field employ the Encrypt() and Decrypt() functions to prevent certain modifications. (This is only a side note of this article.)

The other hidden form field I mentioned (WddxData) is empty since upon form submission (after all changes to the ini file data have been made on the client side) it will be filled with the serialized WDDX data that's holding the modified ini file's content. That way the whole WDDX package is submitted back to the server in order to update the ini file.

Let's take a closer look at the JavaScript functions in main.cfm that handle the client-sided data processing of the ini file's content. As the user interface of main.cfm consists of two single select boxes as well as a text input box, we need a way to dynamically add and remove the entries of a select box. This is possible with JavaScript through the creation of Option objects within the select list. I decided to encapsulate this general logic into a small JavaScript function:

```
// General function to add an <option>
entry to a listbox
function addEntryToList(where, txtValue,
txtDisplay)
{
 tmpNewEntry = new Option(txtDisplay);
 where.options[where.length] = tmpNewEntry;
 where.options[where.length-1].value =
txtValue;
}
```

Following are three JavaScript functions – initSections(), initEntries(sectionIndex), and initValue(entryIndex) – that dynamically fill the select boxes and the text box with the sections, entries, and values taken from the transferred variable ProfilesArray (see above for the WDDX transfer from CFML to JavaScript). The nice feature is that initSections() calls initEntries() with the appropriate parameter, and initEntries() calls the necessary initValue() for the current selection. By default all select boxes and the text box are empty; they're filled by the onLoad-eventhandler of the <body> tag called initSections(). Also both select boxes call initEntries() and initValue() respectively in their onChange eventhandler, which results in the dynamic display of the ini file's sections. That's all for the displaying of the ini file's content on the client. Now the user can browse through all sections of the file and display all the entries without having to go back to the server. This is achieved through the JavaScript ProfilesArray variable that holds all the file's data (here the ini profile) as a model.

How can the user change this data model on the client side by changing values and/or adding new entries or sections? This all happens through the next three JavaScript functions – makeNewSection(), makeNewEntry(), and updateValue(). These functions are called by the "New Section" button, the "New Entry" button, and the "Update Value" button, respectively. Both makeNewSection() and makeNewEntry() prompt the user for the name of the section or entry to add and create the appropriate structure within

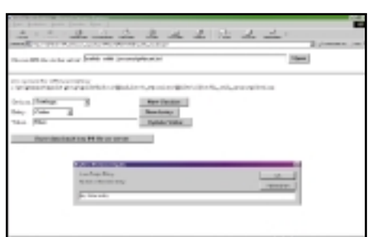


FIGURE 2: Entering the name of a new entry

the client-sided data model, ProfilesArray, and refresh the user interface using the above mentioned init-methods. The updateValue() function simply takes the current value out of the value textbox and stores it in the ProfilesArray using the current selection of the section- and entry-select boxes to reference the position to update. Using these three simple functions it's possible to completely change or add data to the data model on the client without the need for further round-trips to the server. Note that for simplicity reasons, I decided not to introduce the offline deletion of entries or sections, since deleting them from the ProfilesArray doesn't delete them on the server side after submit. To get this feature, each entry in ProfilesArray should get a flag set to "delete me upon server-sided processing" when the user deletes it.

So far we're able to view all the content of the file offline and even add or change data. However, this is only done at the client side! To commit all your changes back to the server (within a single request for all the data), you'll have to push the "Store data back into ini file on server" button that invokes the submitToServer() function. This JavaScript function is responsible for serializing the client-sided data model (ProfilesArray) back into a WDDX package and placing it into the hidden form field, "WddxData", before submitting the form.

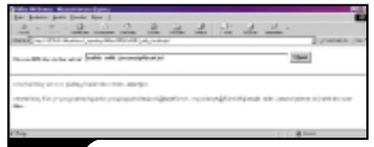


FIGURE 3: Updating all changed data on the server at once

```
// Serializes data and submits it to server
function submitToServer() {
 fr = document.formIniBrowser;
 // serialize the ProfilesArray into WDDX
 myWddxSerializer = new WddxSerializer();
 tmpWddxPackage = myWddxSerializer.serialize
(ProfilesArray);
 // place WDDX package into hidden form
field for submission to server
 fr.WddxData.value = tmpWddxPackage; //
alert(tmpWddxPackage);
 alert("The complete data was serialized into
WDDX and will be submitted to the server");
 fr.submit();
}
```

As you can see from the code, the WDDX serialization of any JavaScript variable is done by first instantiating a WddxSerializer object, then using its serialize(variableToSerialize) method to get the resulting WDDX package. This object is available in JavaScript after the prior import of the WDDX SDK for JavaScript (see above). After placing the WDDX package into the hidden form field, the form gets submitted to update.cfm, which handles the physical updating of the ini file on the server.

After the form has been submitted to update.cfm, this file simply takes the WDDX package out of FORM.WddxData, deserializes it back into the CFML array of structures, and loops over them to update each individual entry in the ini file using ColdFusion's SetProfileString() function.

Voilà, all data has been updated at once! This technique is very efficient and convenient when dealing with dynamic data on the client. If the developer wasn't using WDDX when dealing with large dynamic data updates on the client side, he or she had to write it as a step-based set of templates with round-trips to the server for each request, or deal with dynamic form fields with dynamic names being submitted, which caused a lot of overhead and confusion when using constructs such as <cfset theNewValue = Evaluate("FORM.#someDynamicFormFieldName#")>.

About the Author

Christian Schneider is a ColdFusion developer with over three years of intensive experience in developing CF-based intranet applications for banks and logistic corporations.

mail@Christian-Schneider.de

In this article we describe how combining WML, ColdFusion, and the Fusebox methodology enables wireless data updates of the database of a working e-commerce Web site. To provide some context, we begin with a discussion of what this Web site is and what it does.

NetLook, Inc., is a client of our company, Applied Digital Solutions' Oracle Practice. Its Web site (www.netlook.com) is designed to advertise used cars and help people find jobs (see Figures 1 and 2). NetLook lives in a cluster environment of two NT servers, each running IIS, and ColdFusion Server. These servers also hold the file system that contains auto, job, and customer files (e.g., auto pictures and customer logos).

Heavily involved with multimedia, NetLook presents information via a dedicated cable television station and weekly hard copy publications, as well as over the Internet.

One of NetLook's main features is that the job- and car-related data for these three communication channels comes from a single Oracle database. For the Internet, ColdFusion Server reads the information from the database and then displays it on the NetLook Web site.

A two-step process generates two hard-copy publications, one devoted to automobile sales, the other to job ads. First, a ColdFusion program extracts all the data (car pictures, dealer logos, and car and dealer information) from the file system and the database. Quark then uses this information to automatically generate the magazines.

Producing the TV-based information also involves a two-step process. As in the case with the hard copy, a ColdFusion program first extracts the information from the file system and database, then a NetLook AV specialist synthesizes this information for production into a televised broadcast.

The Role of ColdFusion

Visitors to the NetLook Web site access the database through a front-end written in ColdFusion. To modify the data, NetLook employees also access the database through a ColdFusion-based user interface. They enter data and photographs into this interface, then upload them. In the case of used cars, NetLook provides a marketplace for dealers as well as for individuals. An auto dealership can have its own Web page on the NetLook Web site. To accomplish this, NetLook creates frame-based HTML code for each dealer.

A dealer can select from many types of information (e.g., financing, contacts, inventory) and then display that information on his or her own site. The frame-based architecture enables the dealers to upload their own HTML pages, have those pages appear in sections of the screen, and maintain the navigation across them.

Dealers, of course, often have to modify their own NetLook-based data to keep it current, and ColdFusion is also the basis for the user interface through which they do this.

NetLook and the World of Wireless

With the increasing emphasis on wireless technology and the emergence of WAP and WML, it's possible to add the wireless data update dimension to the NetLook database. Specifically, the combination of ColdFusion with WML forms the basis for an exciting new capability – a capability that enables a NetLook employee to use a wireless phone to accomplish data update.

The modified data is then immediately available over the Internet, on NetLook's next cable TV show, and for the next issues of its hard-copy publications.

In the rest of this article we describe the nature and structure of the programming that makes the data update possible via wireless technology. We also discuss Fusebox, the methodology that's the basis for the code. The programming centers around a wireless phone emulator, available from www.phone.com (see Figure 3). The code is easily transposable to cell phones, with minor modifications to adapt to specific phones.

Working with WML: Concerns and Constraints

Dubbed a deck, a WML program consists of cards. Each card corresponds roughly to one screen, although device-centric display constraints sometimes entail additional screens for the displayed information.

Code that might encompass multiple HTML files can often appear in a single WML file that consists of many cards. Nevertheless, download constraints typically limit the size of a WML file. It's also the case that all cards within a deck are downloaded at the same time. This sets up another type of constraint: if one card initiates server-side processing based on user input to a display generated by a different card, these two cards must reside in separate decks.

How is WML processed? Wireless Internet providers typically offer a service that translates WML requests into HTTP requests. Each HTTP request then goes to a ColdFusion/IIS server, which processes the request, then a ColdFusion program generates a WML file and sends it back to the end user's wireless device. Both the device-centric and download constraints we mentioned before pertain to this generated WML file, not to the ColdFusion program that created it.

Fusebox: Structured Programming for a Stateless World

A hand-in-glove fit with WML, the Fusebox methodology forms the framework for the wireless NetLook programming. Similar to structured programming of several decades ago (as typified by the Modula-2 programming language, for example), this methodology sets up a main program whose explicitly delineated components interact with outside programs (i.e., modules). As with structured programming, a main objective of Fusebox is to enable multiprogrammer teams to work seamlessly on large projects.

Fusebox is different from earlier structured programming techniques – it turns them inside out. In classical structured programming the main program is a sequence of calls to functions and subroutines that carry out the bulk of the processing. In contrast, the Fusebox methodology's main program is a switch statement (denoted by a <cfswitch> tag). Each case in the switch statement (denoted by <cfcase>) carries out a processing module and serves as a target for external calls to the main program. Parameters in an external call determine the case of the switch statement that activates and completes its tasks. Statelessness necessitates this. Because the main program doesn't track or remember its own state, the switch statement provides an easy mechanism for entry into the appropriate point of the main program and for sequencing the processes it carries out.

In ColdFusion, this main program is usually called index.cfm and should handle every request sent to the server. A parameter, usually named *fuseaction*, plays a key role in helping the server figure out what the user needs. Typically, the fuseaction contains values such as "login", "logout", or "save_info". The list of values for this parameter should encompass all the different actions the user is allowed to perform. Additional parameters may be needed to complete the request. When using the fusebox methodology to write an application, the programmer must make sure that the fuseaction values handle all the use cases. (For more information on this methodology, visit www.fusebox.org.)

Conventions

Naming conventions are extremely important in our framework. They make it easy to follow how filenames for cards are constructed, how one card's code references another card, and how a card references decks. Knowing our conventions will also help you follow Listing 1. (Listings for this article can be found on the ***CFDJ*** Web site, www.coldfusionjournal.com.)

Each file whose name begins with dsp_ contains the WML for one card. If the card requires SQL to load the relevant information to the page, the SQL must be included.

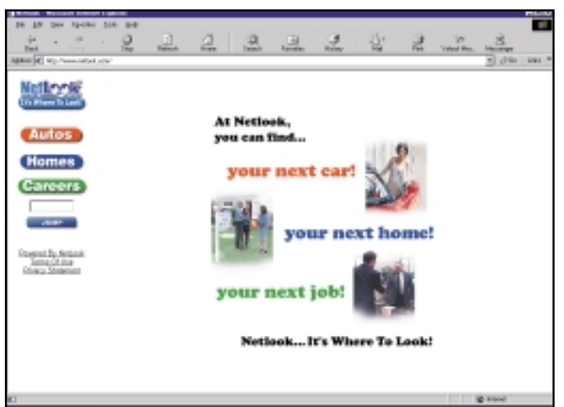


FIGURE 1: The NetLook home page

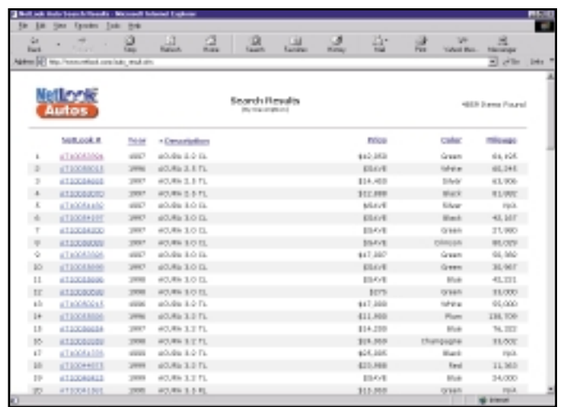


FIGURE 2: NetLook data screen

CUSTOM TAGS FOR MENUS AND MENU ITEMS

A menu custom tag and a menu_item custom tag, both created in ColdFusion, provide a convenient way to build a menu on the fly. In Listing 1, the menu tag builds a menu that consists of each nested menu item call in the third case statement (<cfcase value = "login">).

The menu tag (shown in Listing 3) accepts three attributes. The first attribute, card_name, contains the ID for the card that displays the menu. The second and third attributes, first_include_card and first_card _href, specify the values for the deck's first card in case the menu is not the first card in the deck. This is very useful for our application because it allows us to display the "Select a Dealer" card before we display the menu.

Prioritizing this way is important for Wireless NetLook. As we mentioned in the section "Wireless NetLook: How It All Works," all the items in the menu work from the premise that the user has selected a dealer. Therefore the user must select a dealer before he or she selects anything from the menu. One of the menu items ("Change Dealer") also calls the "Select a Dealer" card, however. Accordingly, the "Select a Dealer" card and menu card are in the same deck. Thanks to first_include_card and first_card_href, everything displays when it's supposed to.

Within the menu, the menu_item tag (see Listing 4) allows the programmer to define each menu selection. Three mandatory attributes – label, href, and action_value – indicate the label to use for the selection, the information to display once the menu item is selected, and the value to assign to a variable so that other programs can track the request. An additional attribute (include_card_href) specifies where to resume program flow after the card selected from the menu item is displayed.

How does all this work? During the start mode of the menu custom tag, if first_include_card is specified, that card is included. During the end mode, the tag loops through all the attributes specified in the menu_item tag. For each menu_item call, the tag builds a list that contains the information for the cards that must be included. When the list is complete, the tag loops through the list and builds and includes those cards. The screen on the phone emulator in Figure 3 shows the menu for this application.

The menu custom tag is handy because it allows us to easily add more items to a menu – something we strongly anticipate having to do as this application's scope expands.

To clarify the discussion, imagine two cards. Each card in a deck must have a card ID, so let's say the IDs for these cards are CardID1 and CardID2.

Our convention uses the card ID in the filename by prefixing it with dsp_ and adding a .cfm extension. So the name of the first card's file is dsp_CardID1.cfm and the name of the second one's file is dsp_CardID2.cfm. Listing 2 shows one of our dsp_ files, dsp_message.

Suppose CardID1's code has to reference CardID2. In a do statement within CardID1 a variable suffixed with _href gets that done. By our convention, the name of that variable will be CardID1_href, and the value it holds will be #CardID2.

An _href-suffixed variable can also handle a reference to a deck. If CardID1's code has to reference a deck, the variable CardID1_href appears in a do statement and its value is index.cfm?fuseaction = deckaction, where deckaction refers to the fuseaction of the referenced deck (i.e., to the use case that the deck's fuseaction fulfills).

Two more conventions and then we're done:

1. ***Act_ file:*** Performs an action on the server side, such as updating the database or setting a variable. These files are strictly ColdFusion programs.
2. ***App_ file:*** Contains code used by every program in an application. This type of module might contain global variables and security checks.

Putting It All Together

Each time the user submits a page a new request is generated for the index.cfm page and a specific value for the fuseaction parameter is passed.

The fuseaction value refers to one of the cfcase statements, which then executes. Each cfcase statement holds a series of cfset and cfinclude statements. The cfset statements define the value of the variables used within the templates that are included in the cfinclude commands.

Each cfcase may refer to zero, one, or more dsp_ files. If no dsp_ files are included, the request is processed in the background. An example is <cfcase value="get _auto_ info">. This particular cfcase retrieves information from the database about a particular car and initializes WML variables.

A cfcase can refer to a single dsp_ file for two reasons. First, only one dsp_ file may be included if the card displays information that depends on user input to a previous card. Second, the size of the card's content may create a download constraint, leaving no memory available to download additional cards.

In our index.cfm file (see Listing 1), <cfcase expression="auto_info_deck_3"> provides an example of both possibilities. This code displays all the models for the particular make that the user entered via a previous card. Because the model depends on the make, these two cards must reside in separate cfcase statements. Also, the list of models might be very long, leaving no room to download other cards.

Two or more cards might be in the same deck if the cards' contents leave enough space to download more than one. Typically, however, it's best to keep them in separate dsp_ files to maintain flexibility. If it becomes necessary to shuffle cards, it's easier to do it if they're in separate files.

Wireless NetLook: How It Works

Listing 1 shows the main program, index.cfm. One overall point about the code further reinforces Fusebox's value: index.cfm contains business logic and is written in ColdFusion. The WML resides in other files (e.g., in ColdFusion modules), is interspersed with ColdFusion, and performs actions such as information display and server-side processing. Thus Fusebox separates business logic from the rest of the code. If the logic changes, major surgery isn't necessary to modify the program.

The first three cases in the switch statement deal with login and session monitoring. The fourth case confirms login and displays a menu for the end user.

The next case – the one that begins with <cfcase value = "menu"> – responds to the user's menu selections. The case after that (<cfcase value = "sel_customer">) allows the user to select the auto dealership in which he or she will update auto information.

These two cases work together in a special way. The menu items work from the premise that the user has selected a dealer. The normal sequence, then, is for the user to select a dealership before the menu displays. Sometimes, during the course of a session, the user has to update car information from a different dealer so the "sel_customer" case must come into play again. For this reason the "menu" case and the "sel_customer" case generate the same cards. The net effect (pardon the pun) is to eliminate repeated downloads of the menu. To understand menu creation and operation in greater detail, see the sidebar "Custom Tags for Menus and Menu Items."

In either case, the user enters a VIN (vehicle identification number) for a particular automobile. If that VIN is in the database, the next case (<cfcase value = "get_auto_info">) populates onscreen fields with appropriate values for the car that corresponds to the VIN. If that VIN isn't in the database, obviously the user must supply values for these fields.

How does the user modify or supply those values? The next seven cases, in which

?

"value" ranges from "autodeck1" through "autodeck7", deal with the user's entries that update the fields. Each case displays information, gets a piece of data update-related input from the user, and uploads to the WAP server. After each upload, the WAP server generates an HTTP request and sends it to the ColdFusion/IIS Server, where a ColdFusion program generates WML that goes back to the wireless device. The code sent back to the wireless device specifies that the next case should activate, get user input, repeat the process, and further augment the parameter list. As the program progresses, each case of the switch statement adds a parameter to a list of parameters.

When the user has finished updating, the last of these case statements sends the fully built list of parameters to the ColdFusion server. Then the next case (<cfcase value = "save_auto">) runs SQL code that uses that list to update the database.

Wrapping Up

The wireless work we described is a significant step forward for NetLook and our company. It paves the way for NetLook to reach its customers and users in an important new dimension and sets up our work for a variety of clients in the wireless future. Experiments that use this code in conjunction with a NexTel phone have been successful.

FIGURE 3: The wireless phone emulator

We feel that Fusebox represents a significant portion of this effort. This methodology has enabled the creation of an architecture that positions the business logic of the system in a central block of code and puts the nuts and bolts of the processing in supplementary modules.

In addition to the obvious facilitation of development and maintenance, this type of architecture holds another important advantage: when bandwidth increases (as it always does!), it'll be possible to take advantage of the increased capability by adding more cards to a deck. This will facilitate downloading and display without tearing apart the fundamental structure of the program.

About the Authors

Margarita Laplaza de Androuin is a senior consultant at Applied Digital Solutions' Oracle Practice. A graduate of Harvard University, she's the technical lead for NetLook development.

Joseph Schmuller is a senior principal Internet analyst at Applied Digital Solutions' Oracle Practice. He's written three books on computing and is an adjunct professor at the University of North Florida.

margarita.laplaza@adsop.com

joseph.schmuller@adsop.com

CFDJnews

LizardTech Shipping Promotional Bundle

(*Seattle, WA*) – For a limited time, LizardTech Content Server 3.0 can be purchased at an introductory rate bundled with LizardTech's MrSID and DjVu technologies.

Content Server allows businesses or individuals to serve limitless amounts of high-quality MrSID images and DjVu documents online. MrSID reduces high-resolution photographs and images to a fraction of their original size, with no loss in visual quality. DjVu is a scan-to-Web standard that reduces scanned documents 50–100 times smaller than PDFs for express Internet delivery, regardless of bandwidth.

The special bundle price is available until February 28, 2001.

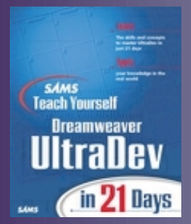
www.lizardtech.com

Dirig Delivers Management for CF Application Server

(*Nashua, NH*) – Dirig Software has released a new Specific Application Manager (SAM) for proactively managing ColdFusion. The new SAM, added to Dirig's SAM portfolio, is designed to enhance management capabilities for IT managers responsible for e-business environments and applications.

www.dirig.com

Sams Releases *Teach Yourself Dreamweaver UltraDev in 21 Days*



(*Indianapolis, IN*) – Sams Publishing has released a book that teaches the fundamentals and advanced features of Dreamweaver UltraDev 4. *Sams Teach Yourself Dreamweaver UltraDev in 21 Days* explains these features in easy-to-follow lessons.

The book, written by John Ray, teaches readers how to create Web sites with personalized dynamic content, develop a browser-based sales and inventory management system, create a discussion forum, and develop a complete e-commerce solution using Dreamweaver UltraDev. The tutorial approach helps the reader learn the basics quickly and then moves on to the more advanced features and concepts.

www. macmillanusa.com

ADVERTISER INDEX

ADVERTISER	URL	PH	PG
ABLECOMMERCE	WWW.ABLECOMMERCE.COM	360.253.4142	2
ACTIVEPDF	WWW.ACTIVEPDF.COM	888.389.1653	27
ADHOST	WWW.ADHOST.COM	888.234.6781	43
ALLAIRE CORPORATION	WWW.VUE.COM/ALLAIRE	877.460.8679	29
ALLAIRE CORPORATION	WWW.ALLAIRE.COM	888.939.2545	35, 51
ANDREWS TECHNOLOGY	WWW.ANDREWSTECHNOLOGY.COM	888.393.0159	31
CAREER OPPORTUNITIES	CAREERS.SYS-CON.COM		56
CFXHOSTING	WWW.CFXHOSTING.COM	866.239.4678	21
COLDFUSIONJOURNAL.COM	WWW.COLDFUSIONJOURNAL.COM	201.802.3020	49
CONCEPTWARE AG	WWW.CONCEPTWARE.COM	49(0)6196-4732-0	23
CORDA TECHNOLOGIES	WWW.POPCHART.COM	801.802.0800	11
CYSCAPE	WWW.CYSCAPE.COM	800.932.6369	39
EVOLUTION B	WWW.EVOLUTIONB.COM/SYNERGYFORFREE	877-622.7551	59
ICJD	WWW.JAVACON2001.COM		47
INTERLAND	WWW.INTERLAND.COM	877.887.3864	3
INTERMEDIA	WWW.INTERMEDIA.NET	800.379.7729	60
JDJ STORE	WWW.JDJSTORE.COM	888.303.JAVA	57
MACROMEDIA	WWW.MACROMEDIA.COM	415.252.2000	4
MISTRAL DESIGN GROUP	WWW.PURPLEHOSTING.COM	214.827.1114	44
NETMORF	WWW.NETMORF.COM	877.533.MORF	15
PACIFIC ONLINE	WWW.PACONLINE.NET	877.503.9870	39
PAPERTHIN	WWW.PAPERTHIN.COM	800.940.3087	17
SITEHOSTING.NET	WWW.WEARECFHOSTING.COM	877.NTHOSTING	31
SYS-CON MEDIA	WWW.SYS-CON.COM	800.533.7111	54
UUNET	WWW.INFO.UU.NET/WEBHOSTING.COM	877.465.7107	9
VERTICALSCOPE(TOPHOST)	WWW.TOPHOSTS.COM	306.546.3778	25
VIRTUALSCAPE	WWW.VIRTUALSCAPE.COM	877.VSCAPE4	13
WIRELESS BUSINESS & TECH	WWW.WIRELESS-MAGAZINE.COM.	201.802.3020	55
XML DEVCON EUROPE	WWW.XMLDEVCONEUROPE2001.COM		45

?