EXPANDED CONFERENCE ISSUE

U.S. $8.99 (Canada $9.99)

ColdFusion Developer's Journal

ColdFusionJournal.com

October, 1999 Volume: 1 Issue: 5

INCLUDED WITH THIS ISSUE
FREE COLDFUSION DEVELOPER'S BONUS CD

SYS-CON PUBLICATIONS

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
PRODUCTION EDITOR CHERYL VAN SISE
ASSOCIATE EDITOR NANCY VALENTINE
PRODUCT REVIEW EDITOR TOM TAULLI
TIPS & TECHNIQUES EDITOR MATT NEWBERRY

WRITERS IN THIS ISSUE
JEREMY ALLAIRE, CHARLES AREHART, MATTHEW S. BOLES,
RAYMOND CAMDEN, MARK CYZYK, ROBERT DIAMOND,
ROBIN E. DILLEY, ED DONOHUE, STEVEN D. DRUCKER,
BENJAMIN ELMORE, BEN FORTA, HAL HELMS,
ANTHONY KRINSKY, ANDREW STOPFORD

SUBSCRIPTIONS
SUBSCRIBE @SYS-CON.COM
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS,
PLEASE SEND YOUR LETTERS TO
SUBSCRIPTION DEPARTMENT.

SUBSCRIPTION HOTLINE 800 513-7111
COVER PRICE $8.99/ISSUE
DOMESTIC $49.99/YR. (6 ISSUES)
CANADA/MEXICO $69.99/YR.
OVERSEAS $99.99/YR
BACK ISSUES $12 EACH

PUBLISHER, PRESIDENT AND CEO FUAT A.KIRCAALI
VICE PRESIDENT, PRODUCTION JIM MORGAN
VICE PRESIDENT, MARKETING CARMEN GONZALEZ
CHIEF FINANCIAL OFFICER IGNACIO ARELLANO
ACCOUNTING MANAGER ELI HOROWITZ
CIRCULATION MANAGER MARY ANN MCBRIDE
ADVERTISING ACCOUNT MANAGER ROBYN FORMA
ADVERTISING ASSISTANT MEGAN RING
GRAPHIC DESIGNER ROBIN GROVES
GRAPHIC DESIGNER ALEX BOTERO
GRAPHIC DESIGN INTERN AARATHI VENKATARAMAN
WEBMASTER BRUNO Y. DECAUDIN
CUSTOMER SERVICE SIAN O'GORMAN
CUSTOMER SERVICE ANN MARIE MILILLO
ONLINE CUSTOMER SERVICE AMANDA MOSKOWITZ

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

COLDFUSION DEVELOPER'S JOURNAL (ISSN #1523-9101)
IS PUBLISHED BIMONTHLY (6 TIMES A YEAR)
FOR $49.99 BY SYS-CON PUBLICATIONS, INC.,
39 E. CENTRAL AVE., PEARL RIVER, NY 10965-2306.
APPLICATION TO MAIL AT PERIODICALS POSTAGE RATES
IS PENDING AT PEARL RIVER, NY 10965
AND ADDITIONAL MAILING OFFICES.

POSTMASTER
SEND ADDRESS CHANGES TO:
COLDFUSION DEVELOPER'S JOURNAL
SYS-CON PUBLICATIONS, INC.
39 E. CENTRAL AVE., PEARL RIVER, NY 10965-2306

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

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

SYS-CON PUBLICATIONS

In the Beginning...

BY **ROBERT DIAMOND**

I was first introduced to ColdFusion almost three years ago when one of the programmers in **SYS-CON's** Web Services Department suggested we implement it in a few different places on one of our Web sites. As the department manager I said, "Sure, sounds good....By the way, what the heck is it?" For the first six months I regretted the "Sure" part immensely. I still knew almost nothing about the workings of ColdFusion or its potential; all I knew was that the programmer somehow managed to keep crashing our server and the Web site was down almost as much as it was up. Thankfully – for the Web site visitors and for me – ColdFusion and my knowledge of it have come a long way since then.

When this programmer left the company, I had to learn ColdFusion. I complained for all of five minutes until I realized what he knew all along – the dynamic possibilities offered by ColdFusion. It was one of the best things that ever happened to me. Looking back on the whole mess now, I'd like to think we've more than made up for our initial problems. Today, ***CFDJ***'s Web site – www.ColdFusionJournal.com – runs on ColdFusion as do most of **SYS-CON's** Web sites.

Exciting improvements are on the way for ***CFDJ***'s Web site and print magazine. A couple of highlights include an upcoming online chat site with scheduled discussions and Q & A sessions led by CF experts; we also have plans to turn ***CFDJ*** into a monthly magazine starting in the first quarter of the upcoming year. That means twice as many issues per year, which I'm very excited about!

Speaking of exciting developments, I'm particularly excited about Allaire's upcoming release of Spectra, which promises to reduce development time and complexity by offering a set of core services for Web developers. I'm sure you'll agree that anything that speeds up the development process and shortens the time to deployment is bound to create a stir when it's ready for release. It can't come soon enough for me.

To whet your appetite, we've got several great articles in this issue about Spectra – what it is and what it can do for you. "Spectra: The New Application Standard" by Anthony Krinsky is a perfect place to start. He covers all of the features, functionality and possibilities that exist in the latest beta release. "Spectra and the Metadata Service" by Raymond Camden tells you about a feature I'm particularly excited about – content indexing. It's very powerful and extremely easy to use and the results are fantastic.

That's not all this issue has to offer, but for the rest – you'll just have to flip ahead. I'm sure you'll enjoy it.

If you've got any questions, comments or suggestions about the magazine or the Web site, I'm always interested in hearing from you and greatly appreciate suggestions. I can be reached at rdiamond@sys-con.com. If you've got something to say, drop me a line. Here's to the future – it's going to be great!

ABOUT THE AUTHOR
Robert Diamond is editor-in-chief of ColdFusion Developer's Journal. *He can be reached at rdiamond@sys-con.com.*

ROBERT@SYS-CON.COM

Expect the **Unexpected**

BY **BEN FORTA**

No one wants to write buggy code, at least no one I choose to know. Bugs are annoying, bugs are embarrassing. And bugs can cost you (and your clients) lots of time and money.

Bugfree code is the ideal all developers strive for – at least should strive for – but it's a lofty goal not easily attained.

To write bugfree code it's important to understand how bugs are introduced. I'd like to explore what I believe to be one of the primary causes for the introduction of bugs into your code:

failure to expect the unexpected. While many of these ideas apply to application development in general, the positioning of this column relates specifically to ColdFusion.

Flow? What Flow?

Applications are designed with a particular program flow in mind. Users start at point A, go to point B, then to point C. Program flow is an important part of any application, and when developers write they anticipate a particular program flow.

Here are some examples.

- A search dialog is displayed, the user enters search criteria, a search is performed and the results are displayed.
- The user browses a product catalog, makes multiple selections with possible confirmations and provides payment information, after which the order is saved and processed.
- A login screen is displayed and the user provides authentication information, which is validated against a database. If authenticated, access is granted; otherwise the login screen is redisplayed.

In each of these examples there's a logical starting point, then a series of steps. The search dialog must be displayed before the search can be performed, users must select items from the product catalog before the order can be processed, and login information must be provided before access can be authenticated.

In traditional application development, programmers had total control over program flow. Users had no way to access screens out of order, nor could they start from a screen other than the one they were supposed to start at. But Web applications, for better or worse, behave differently. As every Web page has a unique address (its URL), it's indeed possible for users to execute pages out of order, just as it's possible for them to start from the wrong page.

How could this occur? There are several ways. Users could bookmark pages directly, search engines might index pages below your root, and newer browsers feature an auto-fill option that completes URLs for users but frequently uses the most recently visited page in the site (often the wrong page).

To understand this better, look at the following code:

```
<!--- Perform search --->
<CFQUERY DATASOURCE="ds"
NAME="search">
 SELECT title FROM books
 WHERE title LIKE '%#FORM.title#%'
</CFQUERY>
<!--- Create page --->
<HTML>
<BODY>
<H1>Search Results</H1>
<UL>
 <!--- Display search results --->
 <CFOUTPUT QUERY="search">
 <LI>#title#
 </CFOUTPUT>
</UL>
</BODY>
</HTML>
```

What's wrong with the above code?

Actually, this is fairly typical ColdFusion code (it's even commented). It performs a simple database search and displays the results. The search itself is driven by a form that contains a field named "title".

And that's what's wrong. The code makes two dangerous assumptions – that the page will be called from a form and that the form contains a field named "title". If either assumption turns out to be incorrect, ColdFusion throws an error because #FORM.title# would refer to a variable that didn't exist.

What's the solution? There are a couple of things you can and should do. The first is always check for the existence of variables before using them, initializing them with default values if needed. The ColdFusion <CFPARAM> tag is very useful for this, and good programming practices demand that every variable be initialized this way so they always exist (either as submitted FORM or URL values, or as locally created variables). If the following code were inserted above the code example, the page would always execute correctly, regardless of how it was invoked and what fields were passed:

```
<CFPARAM NAME="FORM.title"
DEFAULT="">
```

Sometimes you may not want to use default values. For example, in the preceding code you may not want to display the entire contents of the table if the page is executed directly. If a user ends up on this page without having filled in the form to perform the search, you'd want to send them to the search page instead. Here's one way you could do this (assuming the search page was named search.cfm):

```
<!--- Is form field present? --->
<CFIF NOT IsDefined("FORM.title")>
 <!--- Not present, redirect to
search screen --->
 <CFLOCATION URL="search.cfm">
</CFIF>
```

With this snippet at the top of the page your code is now safe. If users try to execute the page directly, you'll programmatically redirect them to where they really should be.

The truth is, every page on your site should be written so that it's safe to execute directly. That way you'll never make assumptions about program flow, and your code won't break when the unexpected occurs.

Are you up for a challenge? How about writing code that uses <CFDIRECTORY> to traverse your code tree to find all CFM pages, then looping through each one, executing it via <CFHTTP> to see which ones throw errors and which work. You could call this code daily (or weekly or monthly) and e-mail yourself the results so you'll know as soon as possible if things might break.

Changes in Site Usage

Another phenomenon that's part of Web site use is the frequency with which that use changes. Programmers anticipate that their site will be used in one way, but, inevitably, users find another way to use it. That's not a bad thing in and of itself, but it does present an interesting problem.

All applications, not just Web-based applications, are designed so that particular parts perform better than others. This is usually because certain parts of the application are more critical and get more use, so the time and effort needed to improve and enhance the site tends to be best spent on these parts.

The application hums along nicely, handling the load thrown at it, and everything works well until those pesky users start using your site in ways you didn't expect. Suddenly the efficient and highly optimized parts are being executed less and less, and the less fine-tuned (I'm being generous) parts start to buckle under the growing load.

This is actually a very common problem. Some of the largest and most impressive sites on the Internet have fallen victim to it.

The good news is that ColdFusion provides you with an invaluable (and frequently overlooked) tool you can use to identify these potential trouble spots. The ColdFusion Administrator (starting in version 4) features a checkbox that, when checked, allows you to log page requests that take longer than a specified amount of time.

Again, the key here is to expect the unexpected and be ready when it occurs. Individual log entries aren't necessarily indicative of a problem, but repeated entries most definitely are. I'd strongly advise all site administrators to enable this option, even on production sites. Determine what the "normal" response time should be (and then pad that number a little) and have ColdFusion log all page requests that take longer. If specific pages start appearing in the log repeatedly, you'll know where to spend your fine-tuning and performance-enhancing efforts.

Up for another challenge? Write a scheduled event that checks this log file daily for new entries. If it finds any, it should e-mail them to you so you'll know about them immediately.

When Bad Things Happen

You've cleaned up your site. You no longer expect specific program flow, nor do you expect specific usage patterns. That's great, but it's not enough.

The other big trouble spot is reliance on other systems and technologies. Whether it's database integration, execution of third-party objects and components, or interaction with other Internet protocols (like HTTP, FTP and NNTP), the more you rely on other systems and technologies, the more room there is for something to go wrong.

Whether it's database failure, the inability to connect to a specific host, or errors and problems in calls resources, they're all *your* problem because to the end user your site is broken. It's not an ideal world out there; bad things do happen and *you'll* be blamed. And you're not going to escape this one anytime soon.

So what do you do? Again, ColdFusion (version 4 or later) provides a tool to respond to these situations – try catch error handling. While full coverage of try catch is beyond the scope of this column (I'll devote an entire column to error handling in the future), here's the basic idea.

Try catch error handling allows you to trap errors in your code, then pick up the processing elsewhere when an error occurs. The basic page layout looks like this:

```
<CFTRY>
 ... page goes here ...
 <CFCATCH>
 ... error processing goes here ...
 </CFCATCH>
</CFTRY>
```

By wrapping an entire page between <CFTRY> and </CFTRY> tags, any error that occurs within that page will be trapped. As soon as an error is caught, processing will be taken over by the catch block (the code between <CFCATCH> and </CFCATCH>). There's no limit to what you can place in the catch block–you can change variables, redirect requests, generate e-mail, write logs and even try to fix error conditions.

Experienced developers take this one step further by nesting try catch error handling. Generic error handling is implemented at the page level, and additional error handling is implemented around specific features or functions. This type of implementation provides the greatest level of control and allows developers to not just *expect* the unexpected, but also to *handle* the unexpected.

Conclusion

The Internet is a highly dynamic and configurable environment. With all that flexibility and power comes the need for greater caution and planning to ensure that applications don't suddenly break. Bugfree code is a wonderful ideal, but for most of us, writing good code that doesn't break in unexpected scenarios is as close as we'll get to attaining it.

The bottom line is that as developers writing code for this new environment, we must all expect the unexpected–because it's going to happen and usually at the least opportune time.

ABOUT THE AUTHOR

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

BEN@FORTA.COM

Its abstraction of **work flows** and **functionality** allows developers to rapidly rework their applications

BY **ANTHONY KRINSKY**

Allaire Spectra is the new standard for application development in ColdFusion.

Not only does this product provide powerful, out-of-the-box content and work flow management tools, but its seven core services constitute a powerful application framework that developers can use in many different and creative ways. Because developers will be coding to the same interface – the Content Object API (COAPI) – carefully written projects will be reusable on any Spectra system. A syndication facility will even allow developers to "beam" their projects to other Spectra servers over HTTP. Such substantial code reuse marks a fundamental turning point in the ColdFusion developer community. As an added bonus, Spectra is a well-coded, open-source application. For many junior (and even senior) programmers, Spectra is a veritable programmers' paradise.

Fully understanding the Allaire Spectra application framework will take you weeks, perhaps months. In this article, I provide a top-level analysis of some of the more important features and normative issues concerning COAPI and Content Object Database.

Evolution

Considering Microsoft's release of Site Server several years ago, Spectra seems like a latecomer. While a simpler product could have been developed earlier, a robust execution would have been impossible without CF 4.0 functionalities such as structures and WDDX. This product isn't a completely new design but was heavily influenced by Allaire's internal, third-generation publishing system, code-named "Willis," which drives Allaire.com. However, its ancestry is almost irrelevant since its functionality eclipses that of its predecessor.

Core Services

The first thing you'll notice about Spectra is its Webtop user environment. The Webtop is a GUI that provides end-user access to content, process, user, site and system tools. Allaire has clustered these functions into five tabs -- Content, Business Center, Site Design, System Design and System Admin – that loosely correlate with the five broad types of participation Allaire calls the "Spectrum of Participants" – Business User, Business Manager, Site Designer, Interactive Developer and System Administrator.

If you use Spectra for a traditional content-managed Web site, you'll do most of your site maintenance and content updates in the Webtop. As the "iBuild" sample application demonstrates, the only thing that happens outside the Webtop is the scheduling of content items to "container" objects you defined and hard-coded into your site's CFML pages. The Webtop is, in effect, the Spectra application.

For a hard-core Spectra developer, however, the Webtop is more of a sample application than a useful destination. To date, the Webtop is the most sophisticated implementation of the Spectra COAPI, the interface that wraps up Spectra's seven core services. These services constitute the Spectra application framework, which will become the foundation for any extensions you write to the basic, out-of-the-box publishing and process management functionality.

Once you start to understand these services, you'll see that applications far more robust (and certainly better looking) than the Webtop are possible. At first, I expect that we'll see major efforts to use the Spectra application framework for subscription content sites, end-user and business-to-business e-commerce sites, and intranet and extranet applications such as help desk, sales-force automation and project management. As the core product evolves, you'll find the range of applications written to Spectra as varied as those written to Lotus Notes or even Microsoft Windows (other application frameworks with which you may be familiar).

The Black Box

Webtop is going to pique your interest about what is going on under the hood. Spectra's XML-hybrid DataStore may be its most defining characteristic. When you load up Spectra, you'll notice that it installs Sybase SQL Anywhere 6.0 and creates an ODBC link to a certain "CFAObjects" database. While the database is accessible to any program that reads ODBC (or SQL Anywhere 6.0), you won't find information about the database in Spectra's 1,000-plus pages of documentation.

This inconvenience isn't an oversight. The Spectra Object Store is not to be fiddled with; it's a "black box" that belongs to the Spectra COAPI. You talk to the COAPI and it talks to the database, so you can do sophisticated things very quickly without

worrying about data corruption, reliability, query optimization or interoperability between Spectra services. You'll find that Microsoft has taken the same approach with Exchange. While providing sophisticated programming APIs for managing messages, calendaring and groupware, Microsoft doesn't provide access to the Exchange message-store itself (purportedly an SQL Server derivative).

While you can "get at" the database, you probably shouldn't try for several reasons. First, there are lots of intricacies with the database. It's very likely that you'll screw something up since object relationships, constraints and dependencies are buried deeply in WDDX packets rather than being clearly defined in database metadata or in Microsoft Access table, field or relationship properties. Second, when you start tinkering, you're giving up customer support and forgoing any significant future upgrades. Allaire has a full team dedicated to tweaking and improving Spectra performance; we should see some dramatic changes in the next year. This scenario illustrates my point: Allaire could rewrite the entire product in C++ or Java (JRun) and swap a native object database (ODBMS) for the relational database (RDBMS) it ships with now. Imagine fully personalized, uncached pages instantiating in 10 milliseconds! Don't hold your breath, but do be mindful of the depth and breadth of enhancements Allaire will be considering. If you've done any serious tinkering with the object store, your data won't upgrade properly and you'll miss out on future advances.

Inside the Box

If restricted access to the DataStore piques your interest, especially if you haven't tried building an object-relational framework yourself, you'll probably enjoy the following discussion about the data architecture.

Consider this hypothetical, pre-Spectra case – we're setting up intranet-application document management and discussion forums. In a normalized database, you'll probably have at least five tables: users, documents, attachments, topics and threads. Next, the boss asks us to provide core services (as Spectra provides) for every element in the system. These services include permissions, logging and metadata. How would you do it?

Figure 1 highlights the difficulties you may encounter trying to "bolt on" tables to handle such services. As tables proliferate and users require more complex queries, the joins and conditional evaluations required in this scenario make it infeasible to maintain.

Figure 2 shows how the addition of a central-objects join table can simplify things substantially. The Objects table provides a convenient "handle" that can be used to access any element in the system.

Allaire could have provided approximately 50% of Spectra's functionality with such an approach. However, they wanted much richer functionality. Spectra needed to offer end-to-end object creation and management through a Web-based GUI without the intervention of a database administrator (DBA). It needed to allow syndication of data and data types (these could include entire applications!) between Spectra services, as well as permit the storage of complex objects that you couldn't easily squeeze into a relational schema – a work-flow prototype or objects with other objects embedded in them, for example. The answer was the innovative XML-hybrid object store shown in Figure 3.

Rather than having tables and columns for each information category in the system, Spectra uses the WDDX to serialize object structures into XML for storage. In addition, Spectra keeps a data dictionary that it uses to bind object properties to rows and columns in external data tables and to a special "properties" table it keeps to facilitate pattern matching on object properties (so Spectra doesn't need to plod through every object's WDDX data packet to find employees whose first-name property is "bob").

The process of defining where object properties will be stored in a relational database is called "object-relational" mapping. When you need to search on a property, Spectra maps it to the properties table. Otherwise, it doesn't even bother, simply serializing the entire object structure and stuffing it in the "object-data" column in the Objects table.

Should you need to store data relationally or simply outside the DataStore, Spectra allows that too. Figure 4 shows a scenario where the developer has created an external table mapping linked by a foreign key to the central object table.

Object Vocabulary

Before you can actually dig your teeth into Spectra, you need to understand its vocabulary, which can be confusing. For starters, you need to understand the core building block of any Spectra application, the "content object."

In more familiar terms, the content object is nothing more than a CF 4.0 structure with keys and values that map to the column names and respective data values of a single row of a database table.

Figure 5 illustrates the concept of mapping a row of data to a structure, stObject, that Spectra understands. Storing data in a structure rather than a query row is convenient; it allows you to easily pass the data between custom tags without iterating through or copying large blocks of data from one place to another.

Building upon this simplified model, there are a few other things that Spectra cares about. First, all identifiers in the system are known by a 35-character Universally (or Globally) Unique Identifier (UUID or GUID). You can make one yourself with the #CreateUUID()# function in ColdFusion 4.01 or later. For example, Jane's UserID would be "C58712BA-268F-11D3-BDFC0000863184AB" rather than "2." UUIDs are important because they facilitate synchronization and syndication. Since identity values (auto-incremented numbers) repeat between tables and databases, it's virtually impossible to check for duplicates and make updates between systems without assigning globally unique identifiers to every element in the system. Under the hood, you'll find that all databases that provide synchronization capabilities use UUIDs.

Second, Spectra has its own way of articulating the "Employee" Table. Rather than giving "Employee" a table of its own in the database, Spectra creates a record in the Types

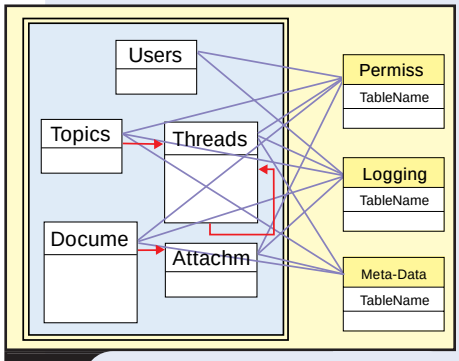


FIGURE 1: A Standard Data Model

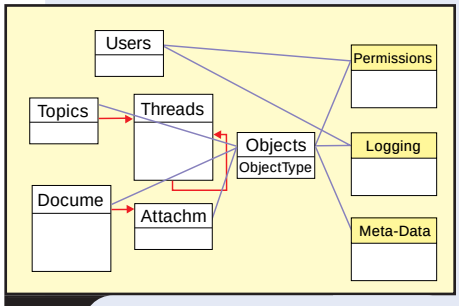


FIGURE 2: A Standard Data Model with a central objects join table

(Object Types) table that contains metadata describing what kinds of information Employee records can or must have. Thus, Spectra calls "employee" an object type. In some object-oriented languages, it would be called a class instead. Webtop is a GUI where you can define object types and also create "instances" of that type. If you prefer a relational database metaphor, Webtop is your data modeling environment. Using our example, "instances" correspond to rows in the table.

Spectra uses object-oriented programming terminology to describe its object types. What Spectra calls properties correspond to column names in the example above. In this case, the properties include FNAME, LNAME and EMAIL. Properties also have UUIDs that allow them to

be reused between object types (e.g., SportsNewsArticle and MetroNewsArticle both have "Title" properties), but we refer to them by their real names.

As discussed earlier, Spectra provides flexibility not available in a relational schema. Through WDDX it permits you to embed objects in other objects and properties in other properties.

In addition to properties that you define,

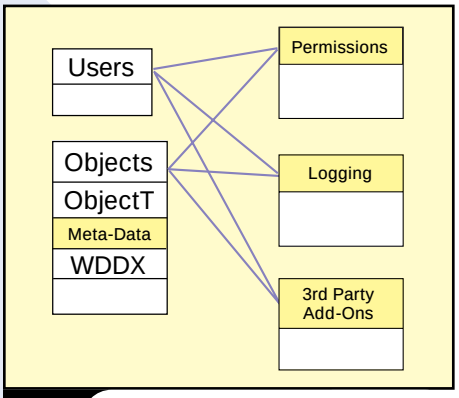


FIGURE 3: Spectra's WDDX-based object store

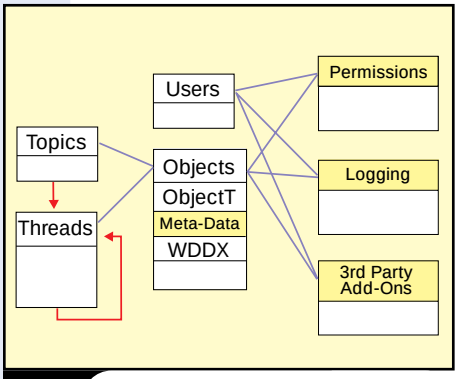


FIGURE 4: Foreign-key mapping

```
stObject=StructNew()
stObject["ObjectTypeID"]="Employee";
```

TABLE: EMPLOYEE

EMPID	FNAME	LNAME	EMAIL
1	John	Doe	john@unknown.com
2	Jane	Doe	jane@unknown.com
3	Tom	Thumb	shorty@unknown.com

```
stObject["ObjectID"]=2
stObject["FName"]="Jane";
stObject["LName"]="Doe";
stObject["Email"]="jane@unknown.com"
```

FIGURE 5: Conceptual underpinnings of the content object, stObject

every object type automatically inherits standard system properties managed internally by Spectra so you don't need to worry about them.

You can also define "methods" in the Webtop administrator that correspond to actions one might want to take with objects. Standard actions automatically assigned include "create," "get," "display," "preview," "edit," "delete," "editmetadata," "editsecurity" and "editattributes."

When you want to execute an object "method" (display me), Spectra asks you to invoke the method for that object with a tag such as cfa_contentObjectInvokeMethod (there are actually several that will get the job done). If you want to display an employee record, specify in the tag that you want to call the "display" method on an object with a certain ObjectID. Spectra checks its data dictionary to see whether you've written a display method "handler" for that object type. Method handlers are simply CFML files that do things with objects (conceivably nothing). A display method will usually CFOUTPUT property keys in the stObject data structure (e.g., <CFOUTPUT>#stObject ["FNAME"]# #stObject ["LNAME"]#</CFOUTPUT>).

If you've written custom handlers, the system CFINCLUDES your handler file, otherwise it looks for a default display method (which outputs a plain vanilla table). The handlers you write determine how your site will look.

The Content Object data structure is one of 17 system-defined data structures Spectra uses. These structures provide a common format for passing around data in the COAPI and allow one component to understand data from another.

COAPI=Maximum Reuse

The basic premise of Rapid Application Development (RAD) is code reuse. If I've already written a tree structure formatting routine, why not wrap it up as a custom tag and call it every time I need to format parent-child nodes into a tree? In another programming language, I might call this a function rather than a custom tag. We use functions and function libraries in many programming languages so we don't need to rewrite code.

To maximize code reuse, it makes sense to break down functionality into as many indivisible, carefully constructed functions as possible. The process of distilling reusable functionality from a specific application is called *abstraction*, which usually refers to the main characteristics of the entity rather than the details. Allaire has abstracted core Spectra functionality into several hundred custom tags that constitute the COAPI. Spectra explicitly divides its custom tag building blocks into three tiers loosely corresponding to tags that do new things (Tier 0), tags that use those tags (Tier 1), and tags that use all other tags (Tier 2).

While there are exceptions, Tier 0 Tags provide Spectra functionality that can't be abstracted any further. For instance, CFA_contentObjectGet actually makes the SQL call to retrieve an object from the database and returns a structure in the attribute r_stObject that can be used in higher-tier tags that render or process object data. Work flow, publishing and site-modeling tags are found in the next level, Tier 1. These tags all reuse Tier 0 Tags to get information in and out of the database and to implement common functions. Tier 2 Tags reuse Tier 1 and Tier 0 Tags. Tier 2 is reserved for application-specific functions that you'll write (e.g., CF_RenderADiscussionForum).

If you closely examine the custom tags that ship with Spectra (certain tags are reserved for internal use, since they could be destructive if used improperly), you'll find about 20% of them can be used outside of the Spectra environment. These tags are completely neutral as to where data comes from and is stored, which is to say they don't reuse any lower-tier tags that reference anything in the object store. Not surprisingly, this collection of tags includes rewrites of many existing functions in the Developer Exchange.

The syntax of COAPI's data structures and the fact that it cares how data is stored differentiates Spectra from application frameworks such as Fusebox or other proprietary coding techniques. It also points to the theoretical limit of code reuse in ColdFusion without agreement on a common vocabulary for describing data and a known data dictionary for describing data types.

You won't find general purpose custom tags in the Developer Exchange that deal with data without a slew of attributes required to describe that data, which is why there are so few of them. Since Spectra keeps an inclusive data dictionary in memory, we never need to pass one around to every data-aware function!

For example, a "display handler" for news articles in a publishing application need not know how or where Spectra stores stories in five different languages. It simply needs to know to reference #stObject("title")# when it wants to render the title property (you might prefer thinking of it as a column) of an object.

An "edit handler" that updates names in the corporate directory will simply set stProperties("firstname")="Tom", and then pass the stProperties structure to CFA_contentObject. Behind the scenes, Spectra consults its data dictionary, finds out how and where "firstname" is stored for the "name" object type, and then takes care of writing the appropriate SQL update statement for you. For that matter, it could be storing the data as a memory structure on the system BIOS chip (of course, that's absurd). It doesn't matter to us; the COAPI is taking care of it so we can focus on building more interesting and capable processing and presenting controls.

Because the COAPI makes it so easy to write data-aware tags and handlers, you'll witness an explosion in the number, variety and complexity of them. Moreover, since your COAPI is the same as the next guy's, you'll witness massive amounts of code reuse within

SIGNIFICANT PRODUCT CHARACTERISTICS

Characteristic	Plus	Minus
COAPI	Faster, easier, less error-prone development. More predictable development time lines. Enables widespread code-reuse.	Substantial learning curve. Reduced control and flexibility. Deeply nested custom tags reduce performance.
Globally Unique Identifiers (GUIDs)	Facilitates synchronization, syndication.	Marginally slower data access. Longer IDs make for heavier pages and JavaScript.
Central Object Table	Allows retrieval of collections containing arbitrary types of information. Facilitates provision of common services such as permissions and logging for all objects.	Complicates data extraction. Does not natively support relational data models, integrity or dependencies.
Object-Relational Mapping	More intuitive, applicable Object-Oriented development. Supports traditional relational databases. Allaire takes care of optimization and query-building.	Mapping incurs substantial performance overhead. Substantial performance penalties for complex functions (can be mitigated through caching).
WDDX Object Storage	Enables persistence of virtually infinitely complex objects. Extremely fast retrieval of complex objects (one "get").	Incurs WDDX deserialization overhead with every "get."
Object-Level Permissions	Context-insensitive retrieval and presentation functions.	Error-prone input. Incredibly slow without permission caching.
"Black Box" DataStore	Data integrity. Data and application portability and syndication. Facilitates upgrades and updates.	Limits customization and optimization. Accentuates product deficiencies (e.g., relationships, synchronization).
GUI System Configuration	Point-and-click data modeling and creation. Rapid system setup and deployment.	Point-and-click data model modification and destruction. Facilitates ad hoc, chaotic development.
Site Modeling Services	RAD site modeling framework.	Limited functionality may not accommodate menuing and layout requirements.

and between projects. As long as you don't reference application or project-specific variables inside your custom tags (make them explicit attributes instead), there's no reason why each and every tag you write in Spectra shouldn't be reusable in another implementation. Using the packaging and syndication tools, installing new functionality may be as simple as typing in a URL. Allaire Developer's Exchange will soon be brimming with Spectra widgets, applications and core-product extensions.

When to Use Spectra

While Spectra is incredibly useful, you're probably not going to want to gut and replace your old apps any time soon. The Spectra architecture was designed to solve certain kinds of problems; it's definitely not appropriate for others.

The table above provides an overview of the trade-offs that you need to consider before and after implementing Spectra. As this article goes to press, Spectra continues to be dogged by poor performance. For high-volume sites, Allaire is suggesting a 4:1 ratio of static versus personalized content. Many developers will find this ratio unacceptable and will need to wait until Spectra is rearchitected or rewritten (in C++ or Java) for better performance.

Growing Pains

Anyone who participated extensively in the Spectra beta testing can attest to the frustration of trying to stick to the COAPI at times. I tried to get multilingual support into the default handlers for seven months and no fewer betas…unsuccessfully! How long will it take for your essential "piece" of functionality to find its way into the COAPI?

However robust, Spectra won't be everything to everyone for awhile, if ever. There's no doubt that if you deploy Spectra aggressively, you'll rewrite some core COAPI tags (e.g., CF_MyBetterContentObjectTag) and wind up with homegrown tags deeply embedded in your application and in unsupported territory.

There are times when you're going to need to slip around, over or through the COAPI. I can only make three suggestions:

1. Don't be too aggressive.
2. Expect some rewriting every time an upgrade comes out.
3. Appreciate the new customizations even if they're the same ones you've spent months coding.

All the books I've read say that frameworks don't become correct until the third go-round. I'm not dissuading you from using Spectra – it's a beautiful piece of work -- but don't expect everything to come up roses the first time.

Allaire the Cannibal?

In the ColdFusion community there are some who may feel that Allaire is stepping on toes. If Allaire spent 25,000 hours (approximately) developing Spectra, just think how many hours could be billed out inventing, reinventing or selling similar functionality!

This argument assumes that clients and employers have unlimited budgets and a limited appetite for application functionality. Clearly the opposite is true: there are an infinite number of things that clients can and want to do on the Web. Spectra allows developers to satisfy more of their clients' needs in less time, increasing client satisfaction, repeat business and/or job security. No matter how Allaire spins it, there'll be no such thing as an open-shut Spectra implementation. Spectra simply creates more viable development opportunities.

There's a good chance that Spectra offers a superior foundation to the one you wanted to build. Most one-off solutions simply don't accommodate the range of new requirements that ColdFusion developers are besieged with every day. Spectra's abstraction of work flows and functionality allows developers to rapidly rework their applications from a solid (and ever-expanding) Spectra code base, rather than having to start from scratch.

Developers' Paradise

On a final note let me urge you to check out Spectra – you'll like the code. Nowhere else in the world will you get open-source access to a panoply of high-quality ColdFusion code (more than 500 custom tags and handlers overall), and I don't think you'll find a more thought-provoking application architecture anywhere. Have you ever tried to build a workflow system? Message queue? A completely abstracted data entry form? How carefully have you implemented error handling in your applications? Do you think in "objects"? Are you using ColdFusion structures enough? Do you implement caching? At multiple levels in your application?

These are some of the questions you'll find answered in the Spectra code base. ColdFusion developers will become much better programmers once they've studied Spectra. Give yourself about a month, full time, and you'll start to understand.

About the Author

Anthony Krinsky is an intranet and extranet application developer with Interactive Documents (www.interactivedocuments.com).

akrinsky@interactivedocuments.com

Server to Client WDDX

Send ColdFusion queries to JavaScript for use in <SELECT> tags

BY
MATTHEW S. BOLES

The Web Distributed Data Exchange (WDDX) is an XML-based technology that allows different Web technologies to exchange data. One function of WDDX is to send ColdFusion queries to JavaScript for use on the client. This article details how to use this functionality with HTML SELECT statements, and assumes you're familiar with ColdFusion and HTML forms but not with WDDX and JavaScript.

WDDX, HTML forms, ColdFusion queries and JavaScript are combined to create a single page providing SELECT statements that are dynamically populated. In addition, one SELECT statement can be filled based on an earlier user selection on the same Web page. In essence, this makes queries that were traditionally available only on the server side now available on the client.

ColdFusion and HTML forms can be used together to dynamically populate SELECT controls from a ColdFusion query. When a user views this form, the query is performed; the SELECT statement contains the latest options from the table that was queried. To enhance this process you may want to populate a second SELECT statement based on a selection the user made from an earlier one. For instance, the first SELECT statement offered options for Colors and Trees (see Figure 1).

If the user selected Colors from the first SELECT control, the second will be populated with the options Green, Orange and Red (see Figure 2).

If the user selected Trees, then the second SELECT will be populated with the options Fir, Hemlock and Pine (see Figure 3).

A common way to implement this behavior is on two separate pages since the first selection wouldn't be known until the user submitted the form. This article shows how to pass the second set of options to the client so the process can take place on a single page.

The first step in implementing this process is the conversion of a ColdFusion query to a JavaScript array (technically, the query is converted into a JavaScript recordset structure). Since you don't know which option a user will select from the first SELECT statement, you'll have to pass all queries that may be needed for options in the second SELECT statement. To do this you'll use WDDX.

WDDX allows you to exchange data between ColdFusion and JavaScript. The following steps are necessary:

1. Perform a ColdFusion query.
2. Use WDDX to convert the query into a JavaScript array.
3. Place the JavaScript array definition in the JavaScript section of the page.

You may then use the data as you would a JavaScript array.

Step 1, performing queries, is familiar to any ColdFusion developer. For this example consider two queries – qryColor and qryTree. Each query reads three rows of data containing a primary key value, a color name or primary key value, and a tree name.

Step 2 uses WDDX to convert the ColdFusion queries to JavaScript usable variables. This is a two-step process, but we'll use a shortcut and make it a one-step process. You need to convert the ColdFusion query into a WDDX packet, then convert the packet into a JavaScript variable. This process uses the <CFWDDX> tag. Without using the shortcut the two lines of code are:


```
<CFWDDX ACTION="CFML2WDDX"
  INPUT="#qryTree#"
  OUTPUT="jsTreeTemp">

<CFWDDX ACTION="WDDX2JS"
  INPUT="#jsTreeTemp#"
  TOPLEVELVARIABLE="jsTreeTLV"
  OUTPUT="jsTree">
```

The first statement uses the attribute ACTION="CFML2WDDX". This converts or, in WDDX lingo, serializes the ColdFusion query into a WDDX packet stored under the name jsTreeTemp. It isn't necessary to understand the actual con-

tents of the packet in order to use it, but, for the curious, see Listing 1 for the query containing the tree primary keys and names converted to WDDX. The indentation used in the packet is for readability only; the actual packet output contains no line breaks.

The second statement uses the ACTION="WDDX2JS" attribute to change the WDDX packet to a JavaScript variable, i.e., deserializes from WDDX to JavaScript. The actual variable type is defined in a file named wddx.js that you'll reference on the page that's installed with ColdFusion. In the code for this page the reference looks like:

```
<SCRIPT TYPE="text/javascript"
  LANGUAGE="JavaScript"
  SRC="/CFIDE/scripts/wddx.js">
</SCRIPT>
```

If this isn't included, you'll probably see an error:

'WddxRecordset' is undefined.

The input for the tag is the WDDX packet created in the first CFWDDX tag. The OUTPUT attribute holds the definition of the JavaScript variable that we'll use in the JavaScript section of the code. The TOPLEVELVARIABLE attribute defines the name of the variable to be used in the JavaScript code.

You need only one tag to perform both these steps using the CFWDDX tag:

```
<CFWDDX ACTION="CFML2JS"
  INPUT="#qryTree#"
  TOPLEVELVARIABLE="jsTreeTLV"
  OUTPUT="jsTree">
```

This converts the ColdFusion query directly into the JavaScript variable and Step 2 is completed.

Step 3 is quite simple. You simply need to start the script section of the page using:

```
  <SCRIPT LANGUAGE="JavaScript">
```

and then use CFOUTPUT to put the JavaScript variable definitions into the script section of the page. When you use the code:

```
  <CFOUTPUT>#jsTree#</CFOUTPUT>
```

you place the JavaScript variable definition on the page. As with the actual format of the WDDX, it's not essential to understand the format of the variable declaration. But again, for the curious, the qryTree query becomes the JavaScript variable defined by:

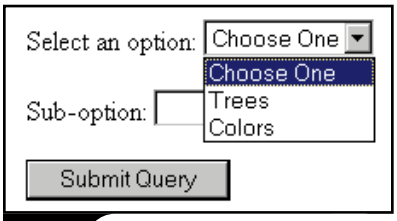


FIGURE 1: The first SELECT statement

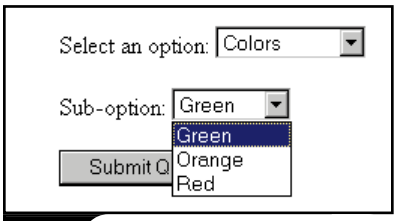


FIGURE 2: The second SELECT with color options

```
  jsTreeTLV=new WddxRecordset();
  _t2=new Array();
_t2[0]=1;_t2[1]=3;_t2[2]=2;
  jsTreeTLV.tree_id=_t2;_t2=new
Array();_t2[0]="Fir";_t2[1]="Hem-
lock";_t2[2]="Pine";

jsTreeTLV.tree_name=_t2;_t0=null;_t1=
null;_t2=null;
```

In the actual code there are no line breaks, just as in the WDDX packet.

To help you understand how to use this definition, think of the query being converted into an array of the general form TopLevelVariableName.columnname[i]. Specifically, the query qryTree generates:

```
  jsTreeTLV.tree_id[0]=1 (the primary
key value) and
jsTreeTLV.tree_name[0]=Fir,
  jsTreeTLV.tree_id[1]=3 (the primary
key value) and
jsTreeTLV.tree_name[1]=Hemlock,
  jsTreeTLV.tree_id[2]=2 (the primary
key value) and
jsTreeTLV.tree_name[2]=Pine.
```

Since JavaScript is case sensitive, it's important to note that in the deserialization process the TOPLEVELVARIABLE attribute value of the CFWDDX tag keeps its case, and the field names from the query become lowercase; the case of the columns from the query doesn't matter. Note that the index for the array starts at 0 and not 1.

With the ColdFusion queries converted into JavaScript variables, it's time to look at the JavaScript that uses these variables.

The goal of the page now is to make these events occur:

1. When the user selects an option from the Select1 form control, a JavaScript function called choosenext() is called.
2. The choosenext() function decides which option the user selected.
3. The choosenext() function fills the DynamicSelect form control with the appropriate query information passed into the JavaScript.

To accomplish Step 1, a SELECT statement is created with the form using HTML:

```
  <SELECT NAME="Select1"
onChange=choosenext()>
```

The statement uses the JavaScript onChange event handler. This means that when a selection is made from the SELECT box, an event occurs (onChange) and the function called choosenext() is called (defined in the SCRIPT section of the page). This happens as soon as the user makes a change in the SELECT statement before the form is even submitted.

The job of the choosenext() function is to populate the SELECT statement that was created with HTML:

```
  <SELECT NAME="DynamicSelect">
    <OPTION
VALUE="">     
    </OPTION>
  </SELECT>
```

Nonbreaking spaces are used because Navigator won't dynamically enlarge the width of the box, but if this isn't done the options won't display correctly .

Now Step 2 needs to be completed, i.e., deciding which selection was made in the Select1 form control. This can be done by using the JavaScript "if" statement in the form.

```
   if
(document.TestForm.Select1.selected
 Index==0)
   if
(document.TestForm.Select1.selected
 Index==1)
```

```
if
(document.TestForm.Select1.selected
 Index==2)
```

This statement takes advantage of the Document Object Model (DOM). The document object is the entire page

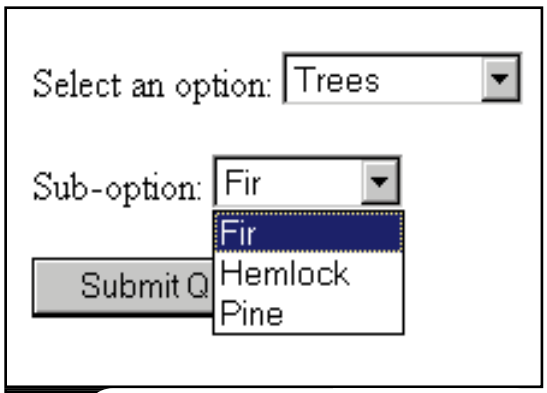


FIGURE 3: The second SELECT with tree options

with the "TestForm" form that contains the "Select1" SELECT control. This object, which can be referenced by the name "document.TestForm.Select1", has a property called selectedIndex. Then it's all put together to see which option the user chose from the SELECT control and guides the program flow into the appropriate section of the choosenext() function.

Now it's time to get Step 3 done: filling the "DynamicSelect" SELECT control with the needed query results (which, as you recall, have been converted into JavaScript variables). First you need to delete any options in the SELECT control by setting the number of options to 0. Again, we need to take advantage of the DOM and use the code:

```
document.TestForm.DynamicSelect.
length=0;
```

This sets "DynamicSelect's" length to 0. All SELECT controls have a length property just as they have a selectedIndex property, which we used earlier.

Next, a loop will be used to put the correct number of options into the SELECT statement. The number of times to loop will be determined by the statement:

```
for (var RowNum=0;
RowNum<jsTreeTLV.tree_id.length;
RowNum++)
```

This creates a loop counter called RowNum that starts at 0 since the index of the array starts at 0. The loop will stop when RowNum exceeds the length of the array, which is equivalent to the number of records read in the ColdFusion query. This number is retrieved from a property of a JavaScript array called length and is referenced by the code:

```
jsTreeTLV.tree_id.length
```

The last part of the loop statement, RowNum++, increments RowNum by one each time through the loop.

Finally, the code that creates the options and puts them into the DynamicSelect form control is used. To accomplish this we use the following lines of code:

```
NewOpt=new Option;
NewOpt.value=jsColorTLV.color_id
[RowNum];
NewOpt.text=jsColorTLV.color_name[Row
Num];
document.TestForm.DynamicSelect.
options[RowNum]=NewOpt;
```

Each time through the loop a new option is created using the JavaScript operator new. This creates a new instance of an option to be placed within the SELECT control. Then the option needs to be given a value for the VALUE attribute and text to be displayed to the user. This is accomplished using the code:

```
NewOpt.value=jsColorTLV.color_id
[RowNum];
NewOpt.text=jsColorTLV.color_name[Row
Num];
```

This code uses the DOM and the value and text properties of the options, which is itself a property of the SELECT statement. Each time through the loop the next values from the JavaScript array are taken and assigned to an option. The last line of code:

```
document.TestForm.DynamicSelect.
options[RowNum]=NewOpt;
```

assigns the new option created to the list of previous options. The options property is actually an array that holds the options. That's why the loop will generate code that will fill the array using options[0], options[1], options[2], etc.

Just before the choosenext() function is exited, the DynamicSelect statement needs to have the first option selected so it'll appear in the list of options. Otherwise a blank option will be used. This is done with the code:

```
document.TestForm.DynamicSelect.
options[0].selected = true;
```

This code makes another reference to the DOM and sets the Boolean property selected (of the options property) to true. This puts the first option in the SELECT control. Remember that the first element of the array has an index of zero.

With the selection of an option from the first SELECT control, the second SELECT control is dynamically populated.

ABOUT THE AUTHOR

Matthew Boles has been doing ColdFusion training for Allaire for nearly two years and has worked with ColdFusion since version 1.5. He is also a certified Novell and Microsoft instructor, although he now devotes his professional time to ColdFusion training and consulting.

MSBOLES@MINDSPRING.COM

LISTING 1

```
<wddxPacket version='0.9'>
 <header></header>
 <data>
 <recordset rowCount='3' fieldNames='TREE_ID,TREE_NAME'>
  <field name='TREE_ID'>
   <number>1</number><number>3</number><number>2</number>
  </field>
  <field name='TREE_NAME'>
<string>Fir</string><string>Hemlock</string><string>Pine
 </string>
  </field>
 </recordset>
 </data>
</wddxPacket>
```

CODE LISTING

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

SPECTRA and the Metadata Service

A way to **index your content** that's powerful and easy

BY **RAYMOND CAMDEN**

Ever wonder why, when you search for something on the Internet, you normally get about a hundred thousand Web sites that are nowhere near what you're looking for? The reason is that many search engines can index only the words of a Web page, not the meaning.

A computer program has no problem reading the text of a Web page – no real problem even discerning the difference between HTML tags and real content. But the meaning of the words is another story. Luckily, we have a way to define a meaning for our Web pages. Traditional use of metadata on the Internet involves the use of META tags within the <HEAD> portion of HTML files. Before publishing your Web site, you go through each file and add a META tag for either the keywords or the basic description of each page. This allows certain search engines to better index your site. For example, AltaVista uses META tags when indexing HTML files. Listing 1 gives an example of this in a typical (but pretty empty) Web page.

At the time this article was written, Allaire Spectra was in its sixth beta cycle. You can expect the screen shots of the Webtop to be slightly different in the current version. Because of this, this article focuses mainly on the GUI tools and less on the actual CFA tags. For more information please read the Release Notes installed with your current version of Spectra.

FIGURE 1: Sample hierarchy

In this example the META tags define a description ("This Web page discusses the *Star Wars* trilogy") and a set of keywords ("Star Wars," "The Empire Strikes Back," "Return of the Jedi," and "The Phantom Menace") that help describe the Web page. This "meta" description allows search engines to better index sites. By providing a set of metadata for each HTML page, the search engine can more easily match up a person's search criteria with what he or she really wants to find.

Working with the Metadata Hierarchy

In Allaire's Spectra product, though, we work with metadata in an almost completely different manner. If you think about it, traditional META tags won't work in a dynamic setting. For example, a Web page that loads a different press release depending on the URL variable passed to it couldn't be adequately described by META tags. You could describe the page as being a press release, but that's too vague to be of any use to a person searching for a particular type of press release.

Spectra tackles this problem by creating a separate entity that represents your Web site: the Metadata Hierarchy. When I say it's separate, I'm not kidding. Spectra contains a completely separate Metadata Hierarchy (and Catalog) object type that we use when working with metadata. Figure 1 shows a sample hierarchy. If *hierarchy* sounds like a fancy or confusing word, just think of it as a tree. In this case the root of our hierarchy is like the root of a tree. Beneath it is a set of categories that act like branches of the hierarchy. In this example I have a category for each *Star Wars* movie and one for books and games.

So how did I create this hierarchy? With any (well-formed) Web site, you normally begin with a site map before slapping together HTML. A site map is an extremely helpful way to describe your Web site before building it. It also helps you map out any problems before doing real work. Just as construction engineers spend many hours working on plans and maps, you don't want to start building a Web site without thinking first about its design. The same goes for our metadata hierarchy. If I were designing a real Web site for a client, we'd get together to map out the various sections and content of the site before building HTML (or CFML) files. In this example let's say I'm building a Web site dedicated to *Star Wars*. I know I'll have one page for each of the movies and a kind of catchall page for information about *Star Wars* books and games. My hierarchy then describes the form of my site before I've even created a single page (or application template). I like to look at Spectra's implementation of metadata as kind of a shadow site. It reflects the nature of the Web site but it's not an exact copy.

Now that I've described a Spectra metadata hierarchy, let's actually build one. Creating a hierarchy is actually pretty easy. All we need is a few lines of code. Listing 2 shows how easy it is.

Don't worry about CFA_GlobalSettings. It just sets up some default variables that our next CFA tag needs. The important tag is CFA_MetadataHierarchyCreate. Our first attribute simply tells the tag what data source to save the hierarchy in. In our example we use the default data source installed with Spectra, CFAObjects. Our next attribute is the label. This creates a friendly way to recognize our hierarchy when we browse content objects in the Webtop. Our last attribute isn't really necessary in this example. The r_ObjectID attribute simply creates a variable that stores the ID of the hierarchy we just made. In our example template we output that ID to the page.

Once we've run this template, we can start building the hierarchy. As you can probably tell, we've created the hierarchy, but right now it's just that, nothing more. Using the tree analogy, we have the seed but no branches or leaves. To add categories we can use another CFA tag or we can build the hierarchy visually. We have a tool, the HierarchyEditor, that we can run from the Webtop.

If you open up the Webtop to the System Design area, you'll see a menu item called Site Categories. Click on the little orange triangle to expand the item, then on Keyword Manager, and you'll see a list of hierarchies like those in Figure 2. The Keyword Manager allows you to edit the hierarchies. If I click on the Star Wars Site hierarchy in the Available Hierarchies list, I can begin work. I start off by selecting the root node of tree (in this case it's simply Star Wars Site) and click the Add Category button. This opens up a JavaScript prompt box that asks for the category name (see Figure 3). If I continue adding other categories, I get a complete hierarchy that should look like my original one in Figure 1. Figure 4 shows the completed hierarchy. (*Note:* Since categories are stored as keys of a structure, we don't remember the order in which they were added. That's why the figure shows categories in a different order than we added them.)

As you've probably guessed from the figures, the hierarchies aren't made up of just categories. Each category can have subcate-

FIGURE 2: Keyword manager

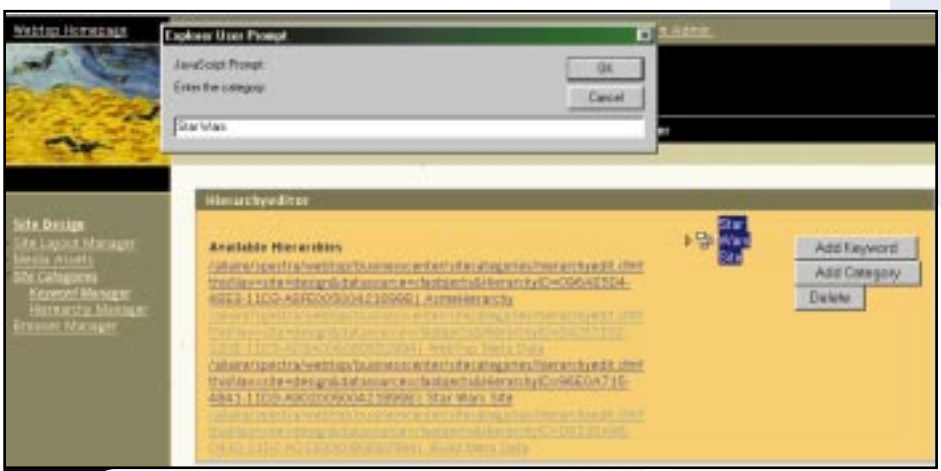


FIGURE 3: JavaScript prompt box

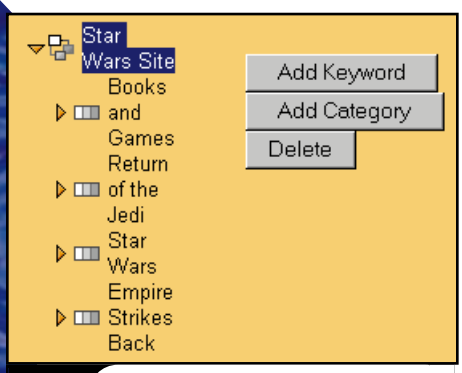


FIGURE 4: Completed hierarchy

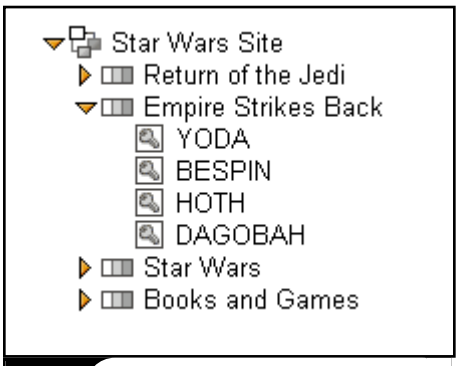


FIGURE 5: Keywords

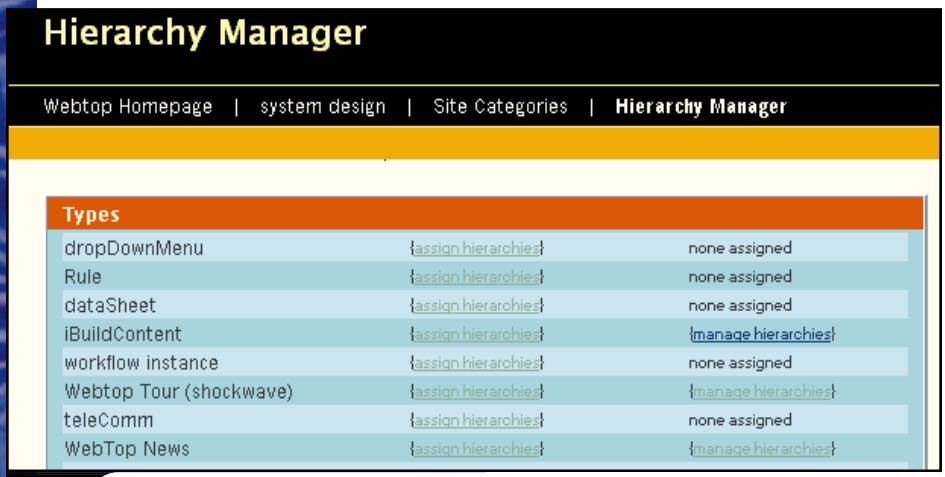


FIGURE 6: Webtop

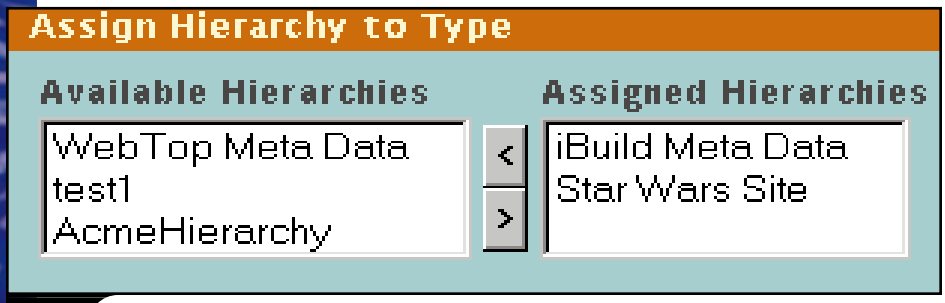


FIGURE 7: Assigned hierarchies

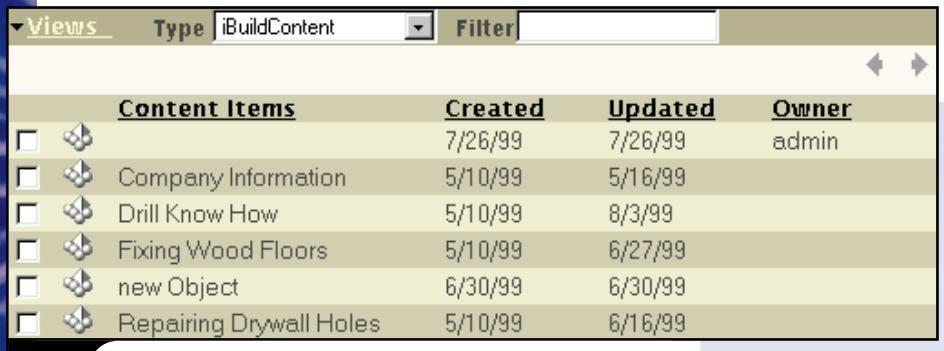


FIGURE 8: Current iBuild objects

gories and keywords. Keywords are the important thing here. They allow us to define a group of words that make up or describe the category in more detail. Using our Star Wars hierarchy as an example, I'd populate the Empire Strikes Back category with keywords that make sense for that particular area, such as "Hoth," "Dagobah," "Yoda" and "Bespin." (As an aside to those of you who recognize that most of those keywords are places in the movie, there's nothing wrong with sharing keywords in multiple categories. I could create a "Places" category to store the names of all the planets visited by the characters in all four movies.) Think of the keywords as the leaves of our tree. They flesh out the categories and give them more meaning. Once again, adding keywords is easy. We can use either CFA tags or the Webtop. I'm going to add the keywords I just described to the Empire Strikes Back category. This time I'll use the Add Keyword button after selecting the ESB category. Once done, I have something that looks like Figure 5. Notice how keywords show up differently than the categories.

Before discussing how we use metadata hierarchies in relation to content, I want to point out that the GUI tools we used weren't required to work with hierarchies. Along with the pretty tree tool, you can also use CFA tags to manually add categories and keywords. These tools aren't discussed here because implementation may change before the final release date. The GUI tools may change in appearance but shouldn't change in usage.

Working with Metadata Hierarchies and Content

Now that we've created a hierarchy, we need a way to tie it to the content of our database. Spectra accomplishes this by assigning keywords to objects. For example, if I assign the Yoda keyword to a content object, any search for the Yoda keyword will match that object. A nice side effect of assigning keywords is that you automatically assign categories in the same process. When I assign Yoda to an object, I'm also assigning the category that Yoda is part of – in this case, Empire Strikes Back.

To assign keywords to objects, once again we have a choice. We can hard-code some CFA tags or use the Webtop. To make it easier we'll use the Webtop. After logging into the Webtop, I return to System Design, click on Site Categories and pick Hierarchy Manager. Once there, we see a list of object types. Figure 6 shows how your Webtop may look.

Since I don't really have a Star Wars site, I'm going to borrow some of the content that Allaire created for the Spectra iBuild demo. I scroll down the list of object types until I find iBuildContent. Before assigning keywords to content, I need to do one extra step. Metadata hierarchies have a concept of "related types" – object types related to the particular hierarchy. For example, if I had a content object type for Star Wars planets, people and spaceships, I'd consider each of these three object types as being related to my Star Wars hierarchy. As before, we have a CFA tag to do this or we can do it in the Webtop. Strictly speaking, there's nothing special about related types. If you want to assign a content object of a type that's not related, there's nothing to stop you. To assign the Star Wars hierarchy, all I need to do is click on "assign

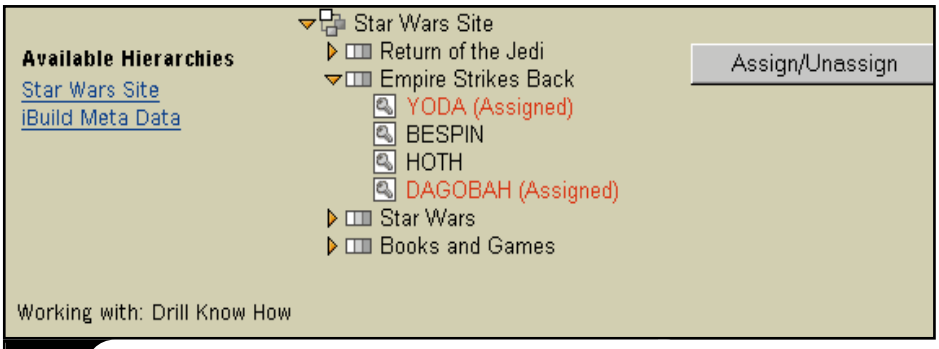


FIGURE 9: Assigned keywords

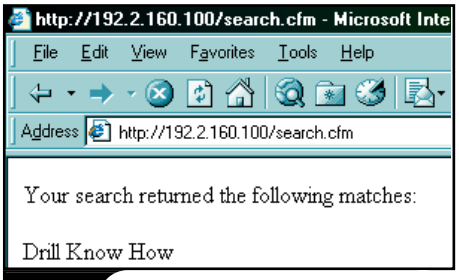


FIGURE 10: Output from keyword search

hierarchies" and add the Star Wars hierarchy (see Figure 7).

Now that we have "related" iBuildContent to our Star Wars Site hierarchy, we can assign keywords to some content objects. Remember when I said that the whole "related types" thing wasn't really necessary? Well, that's true when using CFA tags, but when you use the GUI tool to assign keywords, it actually checks to see what hierarchies are related to the object you're working with. This tool, the MetadataPicker, works with one object at a time. When you load it up, it finds all the hierarchies related to its content type (normally you'll have only one related hierarchy and type) and allows you to assign keywords. To work with the MetadataPicker, I'm going to use the Webtop again. Since I'm working with the actual content, I'll pick the Content section from the top bar of the Webtop. I then pick the Content Finder and select the iBuildContent type. Figure 8 lists current iBuildContent objects. Your list may vary slightly. I can work with one of the objects by clicking on the little pyramid type icon that launches a new window where I can manipulate the object. I pick the "Drill Know How" object and select Categorize. This displays the MetadataPicker. Since we have multiple hierarchies related to iBuildContent, our Star Wars Site hierarchy may not be the default hierarchy. If that's the case, all I have to do is click on the link for that particular hierarchy.

If you remember, we only created keywords under the Empire Strikes Back category, so I want to expand that category. The MetadataPicker works a lot like the HierarchyEditor. I expand the branch of the hierarchy I'm working with (Empire Strikes Back), select the keyword and hit the Assign button. When the page refreshes, I can tell the keywords have been assigned because they show up red and marked as assigned (see Figure 9).

Searching for Keywords

The last step is to actually search for the keywords we just assigned to the content item. Spectra uses two Verity collections to index all metadata assignments: one for keyword assignments and one for category assignments. The MetadataPicker automatically updates the Verity index for us so we don't have to worry about that. If we search for Yoda or Dagobah, we should be able to

find a match. To search the Metadata collections we have two different tags: one searches for particular keywords, the other for categories. Let's keep it simple and just search for a keyword. Listing 3 shows an example search and result display.

Without going into too much detail, the CFA_MetadataKeywordObjectFind tag simply tries to find objects that have been assigned a particular keyword. In this case we know we should get at least one match since we assigned Yoda to one of our iBuildContent items. Figure 10 shows the output of this script. This is only one way of searching for keywords, but I think you can see how incredibly easy it is.

Where Do We Go from Here?

We've only scratched the surface of the Metadata service in Spectra, but I think you can already see that it's a very powerful and easy-to-use method of indexing your content. The power of Metadata, especially in the hierarchy object that Spectra creates, allows you to easily organize your dynamic content and perform quick searches on it. For more information, sign up for the beta (beta@allaire.com) and download the current beta (although at the time of printing Spectra may be released to the public). You'll find it a powerful tool that will forever change how you design ColdFusion applications.

About the Author

Raymond Camden is a senior developer at Creative Internet Solutions (www.creativeis.com), the Web application development firm of Control Data. He is a member of Team Allaire and an Allaire Certified Instructor. Raymond is one of the authors of "Mastering ColdFusion 4" from Sybex.

jedimaster@creativeis.com

LISTING 1: (example of meta tags)

```
<HTML>

<HEAD>
<TITLE>An Example</TITLE>
<META NAME="description" CONTENT="This web page discusses the
Star Wars trilogy.">
<META NAME="keywords" CONTENT="Star Wars,The Empire Strikes
Back,Return of the Jedi,The Phantom Menace ">
</HEAD>

<BODY>
The content of the web page. Lots of text, links, images, etc.
</BODY>
</HTML>
```

LISTING 2: (createHierarchy.cfm)

```
<!--- Initialize some default variables for Spectra --->
<CFA_GlobalSettings>

<!--- Create the hierarchy --->
<CFA_MetadataHierarchyCreate DataSource="CFAObjects" Label="Star
Wars Site" r_ObjectID="ID">
<CFOUTPUT>The hierarchy has been created with the id,
#ID#.</CFOUTPUT>
```

LISTING 3: (search.cfm)

```
<CFA_MetadataKeywordObjectFind DataSource="CFAObjects" Result-
Set="full" Keywords="yoda" r_qObjects="matches">

Your search returned the following matches:
<P>

<CFOUTPUT QUERY="matches">
#title#<BR>
</CFOUTPUT>
```

CODE LISTING

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

Stored Procedures **in Access?** Yes, Indeed!

Get performance, modularization and cross-platform development with a simple change

BY
CHARLES AREHART

While many experienced CF developers will spurn the notion of creating a production CF application with a back-end database in Access, there are far more developers who for one reason or another proceed to – or simply have to – do so.

If you've got to deal with Access, you may as well deal with it in the most effective manner possible. This article introduces a significant new feature for Access developers that will bring dramatic improvements in query processing performance as well as modularization of code. In the bargain, you'll find it significantly eases upsizing from Access, or may enable development of one code base to execute effectively against both Access and an enterprise version of a database.

Bring on the Stored Procedures

One of the greatest benefits of using a "real" database management system such as Oracle, MS SQL Server, Ingres or DB/2 is the ability to leverage "stored procedures" or SPs. An SP is a set of SQL statements stored in the database rather than passed on each request within a CFQUERY tag. Later I'll discuss the many performance benefits of this approach.

To execute such a stored procedure, the traditional way would be to call the named SP in the CFQUERY as follows:

```
<CFQUERY DATASOURCE=…>
 {CALL spname(param1,param2)}
</CFQUERY>
```

Until recently this approach wasn't available to CF developers coding against an Access database because Access doesn't support stored procedures per se. Some developers may have noticed, though, that they could indeed execute Access "queries" using this CALL statement. An Access "query" is more akin to a traditional SQL "view." They're accessible in the Access interface by using the "Queries" tab next to "tables" when viewing a database (see Figure 1).

These "queries" are indeed a stored set of SQL code, but using the {CALL} syntax above, one can't pass parameters to them. CF 4.0 introduces a new means of calling SPs, CFSTOREDPROC and two associated tags. While it's not documented, we've discovered that one can indeed call Access queries as if they were stored procs and can even pass parameters to them. For folks using Access, this is a significant discovery. Allaire recently started documenting this discovery in their Advanced CF Development class.

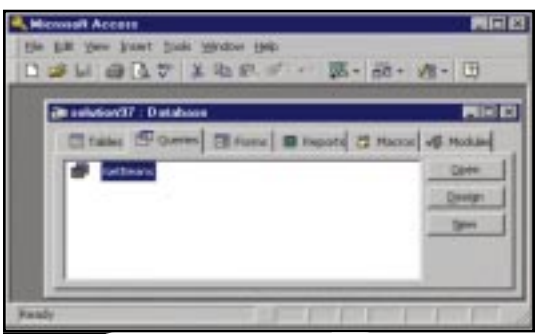


FIGURE 1: The "Queries" tab in Access

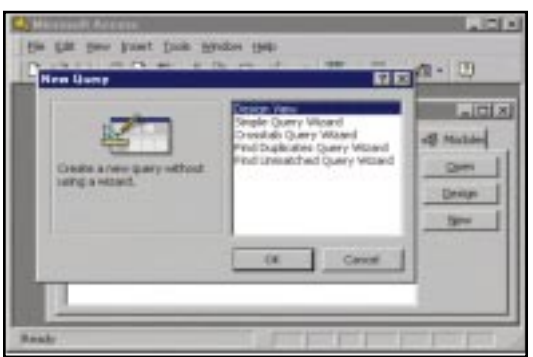


FIGURE 2: Creating a new query

The Benefits of Stored Procedures

Why would any developer, using Access or not, want to consider the SP approach? Again, with SPs, the SQL is stored in the DBMS rather than being passed to the database on every CFQUERY call (often referred to as "pass-through SQL"). Using SPs has several benefits over pass-through SQLs.

Most significant is that the DBMS precompiles the SQL in the SP when the SP is created and reuses that compilation until the SP is changed; pass-through SQL is compiled every time it's executed. Just the reduced compile effort alone can bring dramatic performance improvements in the course of a query that's executed hundreds or thousands of times a day. There's also the minor benefit of reduced network traffic – a call to an SP in a CFQUERY typically requires far fewer characters than spelling out all the SQL in a normal SQL query.

Also important is that in most larger DBMSs the compilation of the SQL in the SP will result in creation of a data-access plan based on indexes and other data characteristics, resulting in faster query execution. A pass-through query doesn't benefit from this optimization. Because of this, most developers choose to use SPs in enterprise DBMSs.

Another benefit is that using SPs to hold SQL is a more modular programming approach, bringing all the attendant benefits of code reuse. This is especially useful if the query can be reused in multiple applications or templates. (Some programmers never consider reuse until they have access to SPs, making this another compelling benefit.)

Finally, in most DBMSs you can also impose security over use and editing of SPs as well as implement version control. (This latter benefit won't translate to Access, but that doesn't diminish the other significant benefits.)

SPs are a hallmark of a professionally developed production applica-

tion, yet Access developers have been basically precluded from using the approach. Now that the CFSTOREDPROC tag can call and pass parameters to an Access query, you merely need to learn how to set up those queries and then call them.

Setting up "SPs" in Access

When you open a database in the native Access interface, several tabs show the tables in the database, etc., as shown previously in Figure 1. To create something similar to stored procedures in Access, you'll want to create what Access calls queries or "parameter queries." These are accessed using the "Queries" tab next to the table tab. Let's walk through creating an Access query. Figure 2 shows the steps used to begin creating the GetBeans query.

Selecting the Queries tab shows any previously defined queries. Using the New button available to the right of the screen, we can create a new query. (The "design" button is used to edit queries, and "open" is used to run them from within Access.) In the list of choices presented next, choose the default "design view" and press OK. When you choose to create or edit a query in Access, it opens a visual query builder similar to that in Studio; the first prompt is to choose the tables to be used in the query.

You can use that feature to build a query, but more likely you'll want to copy in some SQL you're currently using in a CFQUERY. Let's assume you want to do the latter. Close the table selector window to see the display in Figure 3.

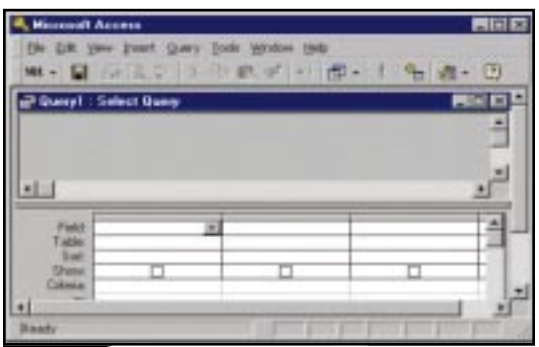


FIGURE 3 The Access query designer

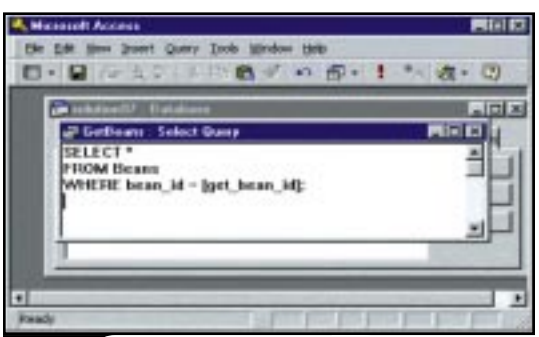


FIGURE 4 "SQL View" of a query with input parameter

Choose "View>SQL View"; you'll be presented with a blank screen on which to build the SQL for your query. Here you can enter SQL by hand or copy it from an existing CFQUERY. If we entered some simple SQL and stopped at that, this query, once saved, could then be called and executed using either the {CALL} approach or the CFSTOREDPROC tag.

If you intend to pass in parameters to dynamically control the processing of the query, you'll need to create what Access refers to as "parameters," then use the CFSTOREDPROC tag to execute the query.

See Figure 4 for an example of such a parameter query. The parameter is enclosed in square brackets (get_bean_id in our example). Normally this parameter is a string of words that, when the query is executed within the native Access interface or related Access programming languages, form a prompt shown to the user. After the user enters a value in response to the prompt, the value is substituted in the query for the parameter. That's interesting, but not what we want at all. Instead, we're going to pass in the value of that parameter on the call to the query in the CFSTOREDPROC (specifically on the CFPROCPARAM). Thus, when naming the parameter, we should choose a single word rather than several words to form a prompt.

Save the query using File>Save and give it a name (following the conventions for naming a table in SQL), which will be the name we use to call it in the CFSTOREDPROC.

Calling SPs (Whether in Access or Other DBMSs)

Now that we have a "query" stored in Access, we can call it with CFSTOREDPROC and its related tags, as shown in Figure 5.

The CFSTOREDPROC tag, usually reserved for use with SPs, names the stored procedure to be executed. We can now use it to execute the Access query. CFSTOREDPROC takes parameters naming the data source and the SP, plus a username and password if appropriate.

The related CFPROCPARAM tag has several attributes naming the parameters to be passed into the SP, indicating their name, value, data type, etc. We can pass in any number of these parameters, with additional CFPROCPARAM tags.

CFPROCRESULT gives a name to the resulting record set, similar to the name we'd normally give to a CFQUERY, so we can refer to the record set later in the template. (This tag also has a "resultset" attribute, used only in DBMSs like Oracle that can return multiple result sets in a single SP call.)

Note that CFPROCPARAM and CFPROCRESULT are subtags to the CFSTOREDPROC tag. In addition, a "type" attribute must be specified in the CFPROCPARAM tag to indicate whether the parameter is being passed data into or out of the SP. The manual says this isn't required and should default to "in," but testing in both 4.0 and 4.01 suggests this isn't true, at least for an ODBC connection to Access.

That's it. Executing the new tags as shown above will result in your being able to process the resulting record set as if you had coded the SQL in a CFQUERY tag. There are several benefits to using this approach, which are reviewed later.

Things to Keep in Mind

While this trick in using Access is nifty, be aware that it doesn't really open up the full range of complex programmatic SQL typically available in full-fledged DBMSs like Oracle or SQL Server. You can't specify any sort of conditional logic in the query, nor does Access support use of temporary tables. You can only perform "simple" SQL, which you could have done in a CFQUERY. (There may be some interesting new SQL opportunities introduced by this approach that we have yet to explore.)

A more important point is that you may find you can't use this approach against an existing Access data source in your environment. If, when you attempt to run a query like the one in Figure 5, you get an error like that shown in Figure 6, you may need to make an alteration to the data source.

Though many may not realize it, there is in fact a set of options that can control the type of SQL actions permitted against a given data source (including insert, update and delete statements, as well as execution of stored procedures). When Access data sources are turned off, they have the option to execute stored procedures by default. You may find that support for stored

```
<CFSTOREDPROC DATASOURCE="FastTrack_Solution" PROCEDURE="GetBeans">
   <CFPROCPARAM TYPE="In" CFSQLTYPE="CF_SQL_INTEGER" DBVARNAME="Get_Bean_ID"
VALUE="1">
   <CFPROCRESULT NAME="GetBean">
</CFSTOREDPROC>

<CFOUTPUT QUERY="GetBean">
   #Bean_Name#<br>
</CFOUTPUT>
```

FIGURE 5: CFSTOREDPROC and related tags

Error Occurred While Processing Request

Error Diagnostic Information

SQL operation unauthorized.

The SQL operation 'execute stored procedure' is unauthorized for the data source 'FastTrack_Solution'. To change SQL authorizations for data sources you should use the ColdFusion Administrator application.

Data Source = "FastTrack_Solution"

SQL = "GetBeans"

FIGURE 6: Default data source definition precludes SPs

procedures has been disabled this way for your data source. You can easily turn that on by editing the data source in the CF administrator and choosing the "CF settings" button, where these choices are available. The way this interface works can be a little confusing: if any of the choices are selected, only the checked statements are allowed. If none are checked, then any sort of SQL statement is allowed. Either check the "stored procedures" option or remove all the other choices, as appropriate to your security needs.

Another challenge is that it can be more difficult to debug SQL errors occurring within stored procedures (using any DBMS). CFQUERY error messages normally show the actual SQL statement in error, but this isn't the case with SPs. (Despite there being a DEBUG parameter on CFSTOREDPROC, it had no effect on this issue in testing.)

There are also some specific issues in using Access queries as SPs. First, when building Access queries, the table or column name can't be parameter driven. In other words, you can't pass in a column name as a parameter to the SP to make the query more dynamic. Second, be aware that testing has shown you can't give a parameter (the item enclosed in brackets) the same name as the column. While that would seem to be a trivial issue, there's a gotcha. It doesn't give an error. Instead, I've found that in a SELECT statement with this mistake, Access simply interprets the parameter as the column name, resulting in the intended criteria being an equality all the time!

That is, if the column were named "bean_id" and the parameter were also named "[bean_id]", any value passed in CFPROCPARAM for bean_id would be ignored. The statement would be compiled as bean_id=bean_id, which evaluates to true. While it doesn't fail, it's as if the criteria weren't there at all. This could be a serious problem if you're not careful.

Finally, you might think that since Access queries are really like SQL views, you could create them using the SQL DDL statement, CREATE VIEW or, better yet, the CREATE PROCEDURE statement. You'd be half right. You can use the CREATE VIEW command (as long as there are no restrictions in the data source definition regarding the SQL that's allowed, as described above). But you can't use the CREATE PROCEDURE command against an Access database.

Benefits of SPs in Access

All that said, the benefits of using this SP approach in developing CF applications against an Access database are substantial and worth the effort.

Improved Performance

Since you're not passing SQL from CF to the database engine, the same query executed as an SP will likely run faster. (Even with CF and the Access .mdb file on the same machine, there's still ODBC communications overhead in even the simplest SQL.)

More important, since the SQL is compiled in the database when the query is saved, there's possibly a substantial reduction in the processing time to execute the query. We've reduced the processing time up to eight times in some queries.

Modularization of Code

You can store common queries that are frequently accessed in the database rather than repeating them within several templates. (It's true that the same benefit could be simulated with a CFINCLUDE holding the SQL, but that wouldn't get the performance benefits above.)

Cross-Platform Development

Finally, using this approach, you can now develop and test an application against an Access database that will be run in production against a SQL Server or a similar database. In most cases the use of stored procedures on a real DBMS will bring significant performance benefits due to the compiling of an access plan, etc. Now the same CF code calling SPs can be run in both platforms (assuming, of course, that you're not leveraging any conditional logic or temp tables that can't be replicated in the Access query).

In a related manner, if you're developing an application that may be deployed on multiple DBMS platforms, you're no longer restricted to suffering the performance pinch of using pass-through SQL on all DB calls. You can code the application to use SPs, and now you know that the benefits of improved performance will translate to all implementations of the application.

It's worth noting, though, that in testing we've found that a very simple SQL statement (perhaps a select returning a single row without a join) executed using the SP approach may actually take just slightly longer than a normal CFQUERY. There is some overhead in executing the SQL as an Access query. However, this overhead is far exceeded by the performance gains on most queries. You may want to test a given query before committing to one approach or the other; the many other benefits of using the SP approach for all your SQL coding may still outweigh any slight performance penalty on a given query.

Now, at least, you know what's possible. Happy coding, and may this be of benefit to all you developers doing production work in Access!

ABOUT THE AUTHOR

Charles Arehart, a senior Coldfusion developer, trainer and researcher, recently joined Fig Leaf Software in Washington, DC. He's worked on enterprise database applications development and administration for 16 years.

CAREHART@FIGLEAF.COM

We're Off to See the Wizard

Developing code wizards in CFML

BY
STEVEN D. DRUCKER
&
ROBIN E. DILLEY

Developing a data-entry interface for managing pesky lookup tables can be a laborious task for even the most seasoned CF developer. These tables usually include a few simple data fields – an identity column for use as a primary key, a text descriptor and perhaps some ancillary information that needs to be managed through a Web-based data-entry interface.

Single template editing (STE) is a methodology for combining insert, update and delete actions in a single .cfm file. While some coding strategies (Fusebox) may increase the number of files required to perform a data-entry task, this method seeks to minimize files by combining functionality.

Once I had optimized the STE algorithm down to the fewest possible lines of code, I created a studio template file to help author these interfaces. Despite using the template, I was still spending an average of 45 minutes developing the customized HTML, CF and stored procedures for a single interface. For applications with many lookup tables to manage, this was a time-consuming and tedious process. After developing my 2,307th lookup-table management file, a thought finally occurred to me.

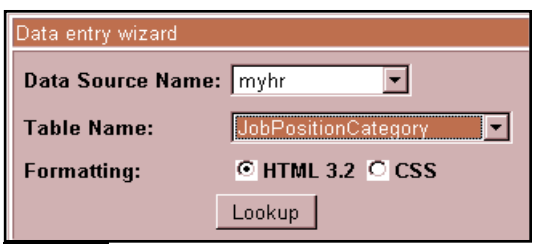


FIGURE 1: **Choosing a data source, table name and code formatting option**

What if I could build a wizard to generate these data-entry interfaces on the fly?

There are several different strategies for accomplishing this task. Optimally our interface would be "point and shoot" – the developer chooses a data source and table name from an automatically generated picklist resulting in a CFML file and accompanying SQL stored procedures. Coding in WIZML, VTOM (Visual Tools Object Model) and ActiveScripting in CF Studio could potentially accomplish the job through the use of several undocumented features yielding programmatic access to the data sources tab. This approach requires the developer to own a copy of CF Studio with an RDS mapping to their CF Server. See the next issue of ***CFDJ*** for more details.

I decided to adopt a more open approach, creating the wizard using CFML itself. The first version of the single template wizard (STW) was developed for SQL Server 6.5. The latest version works for both SQL Server 6.5 and 7.0 as well as giving you the choice of formatting using plain HTML or a Cascading Style Sheet. It can be adapted fairly easily for other SQL platforms. The ultimate goal of this excercise was to reduce the time to code these interfaces to five minutes or less while describing, for our junior developers, a standard methodology for writing code.

Step 1: Use <CFREGISTRY> to get a listing of available server data sources.

The wizard's first step is to select the data source name to work with. Listing 1 illustrates using the <CFREGISTRY> tag to get a list of MS SQL Server data sources from the server. On selection of the data source, a JavaScript function (readtables) is fired to launch tablelookup.cfm in a hidden frame. Tablelookup.cfm uses another SQL Server system stored procedure, sp_tables, to get the table names for the data source. The table names for the selected data source are loaded into the table-name select list. Then you simply need to select plain HTML or CSS as your preferred formatting style. Clicking on the Lookup button kicks off the generation of stored procedures (see Figure 1).

Step 2: Generate stored procedures.

Next, the wizard needed to dynamically create stored procedures for insert, update, delete and select actions on the table. This presented a couple of challenges. First, how would the wizard know which column was the primary key and the data type of each column? We could have made some assumptions based on Fig Leaf's standards for database design. We discovered two system stored procedures in SQL Server – sp_pkeys and sp_columns. When given a table name, sp_pkeys returns the name of its primary key column (very handy in this case); sp_columns returns all information about the columns in the table including data type, length and precision. Armed with this information, dynamically creating the insert, update, delete and select stored procedures was straightforward. Note that a preexisting stored procedure may not be modified; it must first be dropped and then re-created. DROP PROCEDURE syntax, when passed through to SQL Server in a <CFQUERY> block, does the trick (see Listing 2). The STW interface displays four text areas containing the stored procedures for insert, update, delete and select. The procedures are named using Fig Leaf's standard naming convention: insTableName for insert, updTableName for update, delTableName for delete and getTableName for select. Note that the delete procedure generated by the wizard is actually an update action. This is another Fig Leaf standard: to logically delete information from lookup tables instead of physically deleting data, thus creating an audit trail

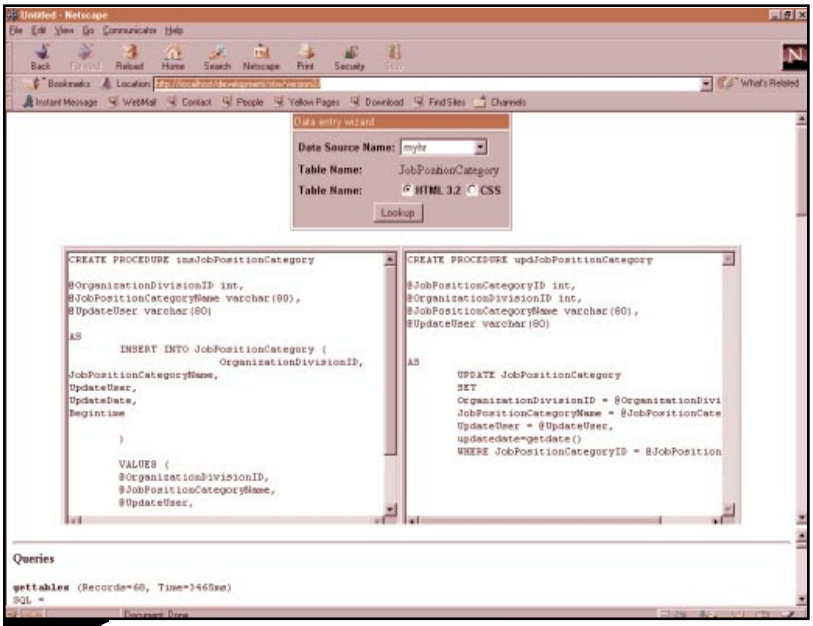


FIGURE 2: CF automatically generates stored procedures for selecting, inserting, updating and deleting information from the specified table.

and freeing us from the chore of unbinding any foreign key references to other tables. Any of the stored procedure code in the text areas can be modified to suit specific needs. Clicking on the Save button saves the stored procedures to your database (see Figure 2).

Step 3: Generate the .CFM file.

After the stored procedures are created in the database, the .cfm template code is dynamically generated and displayed in a text area.

In this step the only required input is the path and file name for the single template file being generated. The Select Directory button fires a JavaScript (myfilebrowser) to open a new window containing the directory tree for the server. The directory tree is displayed through the use of a custom tag developed using the Java file selector tree (ripped from the CF administrator) called CF_fileselector.

One of the challenges in developing the wizard in CFML involved escaping special character sequences that I didn't want CF to process. While ColdFusion provides various methods for performing this task, such as using pairs of double quotes to represent a single double quotation mark (see Listing 3), I found that using the escape sequences made the wizard source code difficult to read and debug.

The table name is passed forward and sp_pkeys and sp_columns are executed again. Using the result set returned from sp_columns, a comma-delimited list is created with the form, prefix and single quotes where the data type is varchar. A series of CFSET statements are used to create variables to contain CF and HTML code used in a single template. The variable mystring contains the CF code found at the beginning of a single template. This code includes the <CFIF> and <CFQUERY> blocks and JavaScript code used to process the insert, update or delete action. Characters recognized by CF, such as the # and single quotes, are escaped in the CFSET statement. These escaped characters are then unescaped, using the Replace function, before the variables are displayed in the text area. The variables mystartformdeclaration and myendformdeclaration contain the beginning <CFFORM> and ending </CFFORM> tags. The variable selectbox contains the select box that contains records returned from the select query, which was executed at the top of the single template. The variable delbutton contains the delete button and the code to dynamically display the button when appropriate. The last two variables, startoutput and endoutput, contain opening <CFOUTPUT> and ending </CFOUTPUT> tags.

FIGURE 3: The application page source file is generated and placed in a text area for further editing.

After creating variables to hold all the content for the single template, they're outputted in a text area named "code". The text area contains the CF and HTML code created by the <CFSET> statements, JavaScript code used by the select box to perform the save action, and HTML table code to lay out the interface (see Figure 3). Depending on your formatting selection in Step 1 of the wizard, plain HTML or CSS syntax is used to format the data entry screen. Finally, the generated code is written out to the Web server's hard disk using a simple call to CFFILE:

```
<CFFILE action="write"
FILE="#form.filename#" OUTPUT="#vari-
ables.mycode#">
```

The STW fulfilled its objective by reducing the time to develop data-entry interfaces to under five minutes. Consider developing your own wizards using the techniques described in this article. The best way to standardize developers' code is to have a wizard that autogenerates the majority of it. Frankly, I'm tired of hearing about what I can do with ColdFusion. It's time we turned our collective attention to what the product can do for us using automated and wizard-based processes. Innovations like Allaire's Spectra prove the feasibility of using CF on a more ambitious scale to autogenerate code. The resulting productivity boost can be staggering.

ABOUT THE AUTHORS

Steven D. Drucker is president and CEO of Fig Leaf Software, an Allaire Premier Partner and Authorized Allaire Training Center, with offices in Washington, DC, and Atlanta, Georgia. An Allaire instructor, Steve started the first ColdFusion Users Group and is a former Team Allaire member.

Robin Dilley is a project team leader at Fig Leaf Software. She has worked with CF since version 2.0 and specializes in leveraging ColdFusion with Sybase Adaptive Server and Microsoft SQL Server.

SDRUCKER@FIGLEAF.COM
RDILLEY@FIGLEAF.COM

Listing 1: Using <CFREGISTRY> to get a list of SQL Server DSNs

```
<cfregistry action="GETALL"

branch="HKEY_LOCAL_MACHINE\Software\ODBC\ODBC.INI\ODBC Data
Sources"
             name="RegQuery"
             type="Any">

<SELECT NAME="datasource" onChange="readtables(this)">
 <OPTION VALUE="">Please Select
 <CFOUTPUT QUERY="regquery">
 <CFIF regquery.VALUE contains "SQL Server" and entry is not
"master">
  <OPTION VALUE="#entry#" <CFIF entry is datasource>SELECT-
ED</cfif>>#entry#
 </CFIF>
 </CFOUTPUT>
</SELECT>
```

Listing 2: Writing stored procedures to the database

```
<!---   check for existence of sp and drop sp if it already
exists  --->
<CFQUERY NAME="checkforsp" DATASOURCE="#datasource#">
 if exists(select * from sysobjects where id =
object_id('dbo.ins#tablename#') and sysstat = 4)
 drop procedure dbo.ins#tablename#

 if exists(select * from sysobjects where id =
object_id('dbo.upd#tablename#') and sysstat = 4)
 drop procedure dbo.upd#tablename#

 if exists(select * from sysobjects where id =
object_id('dbo.del#tablename#') and sysstat = 4)
 drop procedure dbo.del#tablename#

 if exists(select * from sysobjects where id =
object_id('dbo.get#tablename#') and sysstat = 4)
 drop procedure dbo.get#tablename#
</CFQUERY>

<!---   create sp's in database   --->
<CFQUERY NAME="createinsertsp" DATASOURCE="#datasource#">
 #preservesinglequotes(form.createinsertsp)#
</CFQUERY>

<CFQUERY NAME="createupdatesp" DATASOURCE="#datasource#">
 #preservesinglequotes(form.createupdatesp)#
</CFQUERY>

<CFQUERY NAME="createdeletesp" DATASOURCE="#datasource#">
 #preservesinglequotes(form.createdeletesp)#
</CFQUERY>

<CFQUERY NAME="createselectsp" DATASOURCE="#datasource#">
 #preservesinglequotes(form.createselectsp)#
</CFQUERY>
```

Listing 3: Escaping double quotes with carats yielded easier to read code.

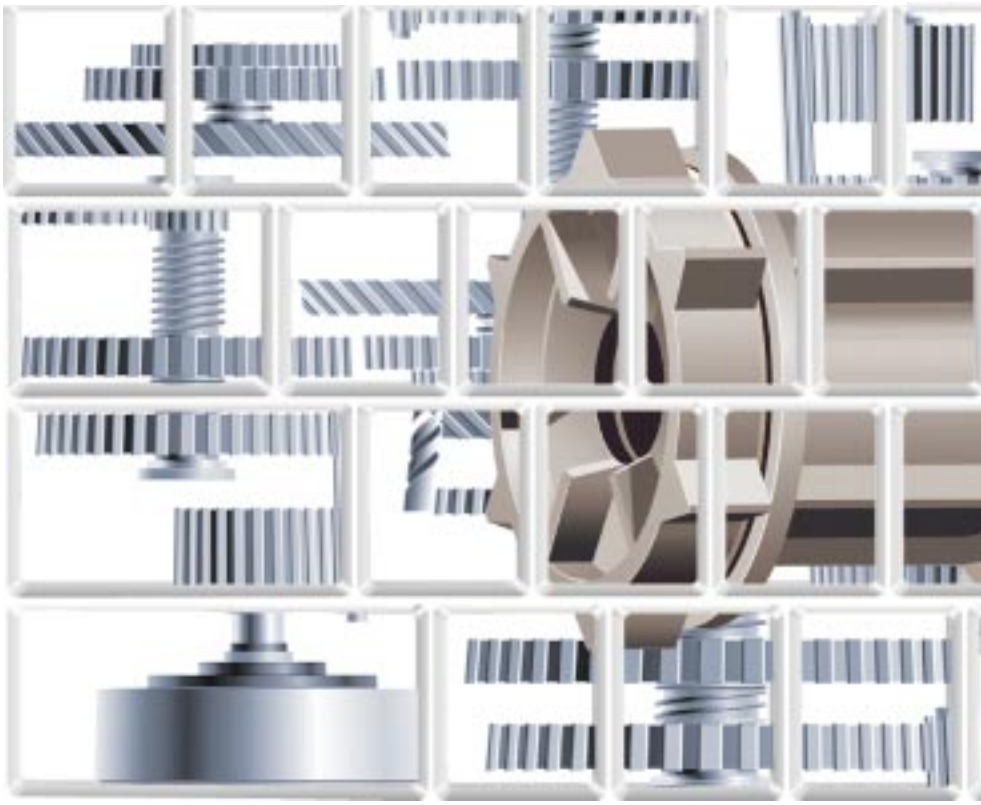
```
<CFIF myaction contains ^Delete^>
 <CFQUERY NAME=^delete^ DATASOURCE=^@application.data-
source@^>
  {call del#tablename# (##my#pkey###,'##session.up-
   dateuser##') }
 </CFQUERY>

 <CFSET my#pkey# = 0>

 <SCRIPT LANGUAGE=^JavaScript^>
  alert(^Record Deleted^)
 </SCRIPT>
</CFIF>
```

CODE LISTING

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

A Help Engine for Web Users

How to Win Friends and Delight Users



BY **HAL HELMS**

The great Victorian novelist Charles Dickens wrote of the time period of the French Revolution: "It was the best of times. It was the worst of times." Or, put more sardonically, historic times are best appreciated by historians. In fact, times of great change are times of great dislocation. Our own historic time, the Age of the Internet, is no exception.

As companies have leaped to embrace the browser as universal client, users have been left with a sense of dislocation. They're being asked to give up their familiar Windows applications for "weblications" that are often hastily built and deployed. For example, we've all become used to the Windows menu bar with "Help" on the far right. When was the last time you saw a Web application with a help system?

We're going to fix that. With this article we're going to create a model that will allow you to easily include context-sensitive help applications with your Web programs using ColdFusion, a touch of JavaScript and your choice of a SQL database. Let's get started.

A Web Help System

Web applications, unlike applications developed for Windows or the Mac, have no standard menu bar with a prescribed place for users to turn for help. This lack of assistance frustrates users and makes the software tool unnecessarily clumsy. Given the lack of standards in the emerging Web world, developers must "roll their own" help systems to prevent this.

A help interface that will be usable across a broad range of application styles needs to be as unobtrusive as possible. I've chosen a simple question mark (see Figure 1) that will be placed in the upper-left corner of all our Web pages. This icon is a standard in our shop; users become familiar with it across different Web applications, comfortable that they have somewhere to turn with questions.

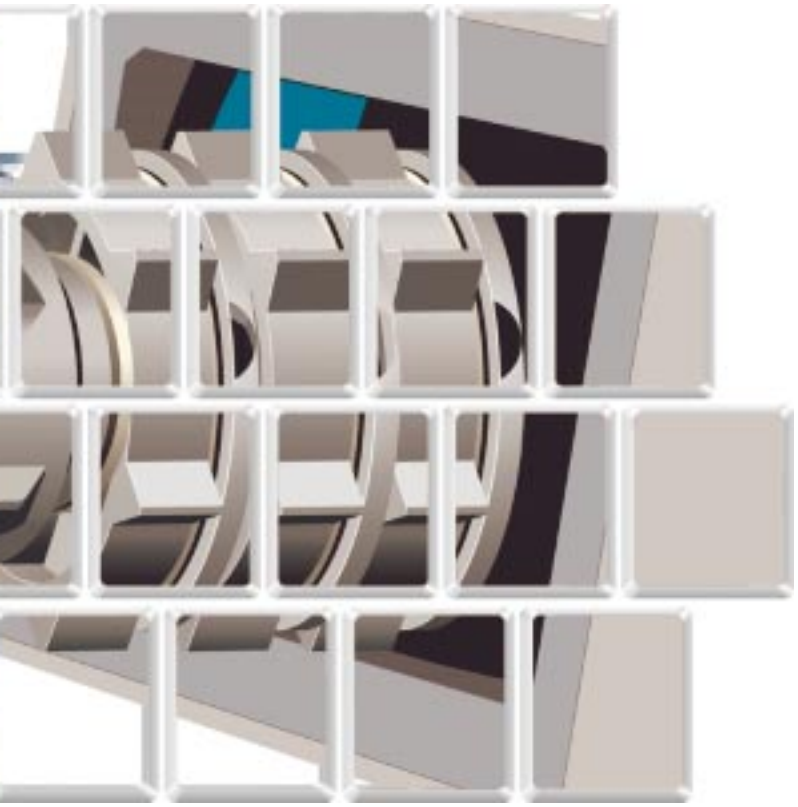
FIGURE 1 Help icon

Clicking on that question mark gives them access to context-sensitive help. Your Web help system can outdo its desktop counterparts by providing users with capabilities they can't get from static help systems. To make our help system more interactive, we placed an e-mail link from the help page to our help staff. If users are confused or need further help with something, they can e-mail a request to the help staff. In our organization these e-mails trigger staff pagers; we want people to use e-mail help as much as possible so it's a priority with us to answer these help calls right away. An added bonus to our help staff is that we find out what page the user was on when they ran into trouble. Help-desk staffers report that this cuts down on the time needed to diagnose the problem.

Further, we can provide users with the ability to add their own notes and tips to the help system. Initially we kept each user's notes private, but were asked by other users to make them available to everyone. Users tell us that this elementary "knowledge base" helps them tremendously. Our help staff likes the ability to "broadcast" new information and hints to users of the application.

Storing Help Information

The basis of any help system, of course, is information. If you're fortunate enough to have a separate help staff, these folks can provide invaluable information about the kinds of help users really need. We have a "help team" look through an early version of a new application, making notes for each page on questions users are likely to have. Often



these notes guide developers in changing the interface to make it more intuitive.

Once this stage is finished, we'll store this information along with some formatting details in a SQL database. We typically use Microsoft Access in the development phase as it's easy to put together and make changes to it. Prior to deployment, we move the table structures to our client's choice of database systems.

We'll need only a single table in our database to hold all the instructions our program will use to find and display help information (see Figure 2). We want our help application to plug easily into many applications; store the name of the page (e.g., index.cfm) as the primary key. (This will allow our help system to read the page name automatically and associate with any help available.)

Pages : Table

Field Name	Data Type	Description
pageID	Text	primary key
pageTitle	Text	appears in title bar
pageDescription	Text	to remind you what page this is
pageHeading	Text	appears on top of page
helpText	Memo	where the beef is
pageBGcolor	Text	bg color
pageBackground	Text	background image
textColor	Text	
linkColor	Text	
vLinkColor	Text	
aLinkColor	Text	
userNotes	Memo	

FIGURE 2: Instruction Table

Separating Application from Help

Our help system needs to (1) determine what page the user needs help with, (2) pull help information associated with that page, and (3) display it in a new "help window." Instead of placing code within the application itself, we create the entire help system independently of the application. In the application.cfm file of the Web application, we place only the following lines of code:

```
application.cfm Code Snippet
<!--- This information is for the help system --->
<cfset application.name = "Team Allaire">
<cfset emailList = "hal.helms@TeamAllaire.com">
<cfinclude template="helpGlobals.cfm">
```

Since the helpGlobals.cfm file is <cfinclude>d in the application.cfm file, it will be called prior to every page. This makes it an ideal place to put generic code that applies to all pages (see Listing 1).

Let's examine it: we begin by creating two variables. The first ("IP") defines the IP address of the Web server the help system will be running on. This allows us to run the system on another machine, separately from the application. The other variable ("application.helpDSN") is the ColdFusion DSN data source.

Next comes the code to spawn a separate help window. As powerful as ColdFusion is, it doesn't handle certain low-level functions such as opening new windows – yet. We need a document scripting language for this. VBScript can handle this, but works only with Microsoft browsers. JavaScript is a better choice in a mixed environment.

The initial <script> tag tells the browser the language to expect. Next, we declare a function called "showHelp" that accepts a single argument called "helpPage." We'll see where it gets this argument shortly. The third line creates a new window that will be 350 pixels by 400 pixels and will display scrollbars if they're needed. The fourth line sets the contents of this window to the URL "#IP#/buildHelp.cfm?pageID= ' + helpPage". We include the parameter that was passed to our function in the query string "pageID=' + helpPage". Our ColdFusion template, buildHelp.cfm, will use this information to determine which help text to show.

The final bit of code on this page checks to see whether any help exists for the current page and displays the appropriate icon. One thing to beware of when dealing with application.cfm files is that they apply to every page – including help pages! Since we don't want a Help icon on a help page, we check to make sure that the current page that's running this code is not a help page.

Earlier I asked you to use the application page name for the help table's primary key. Here's the benefit: a single SQL statement works with all pages in our application. SELECT pageID FROM Pages WHERE pageID = '#GetFileFromPath(cgi.script_name#' will check to see if the current page has any help associated with it. If it doesn't, it will return zero rows and the code <cfif #checkForHelp.RecordCount# EQ 0> will return true.

FIGURE 3: No Help icon

It's worth noting that if it doesn't return true, we place a No Help icon on the page (see Figure 3). We found that when we simply omitted the Help icon when no help was available, users became confused. This simple addition cut down on help desk calls.

What else will happen when helpGlobals.cfm is executed? Nothing. The JavaScript function to fire a new help window exists, but must be called to be activated – by wrapping a normal hypertext link around our Help icon image. Instead of sending the user to another page, the link calls the JavaScript function "showHelp(helpPage)". We send with this request

FIGURE 4: showHelp.cfm page

FIGURE 6: "addNote" action

FIGURE 7: More help page

the pageID information by embedding it into the URL string.

Custom Help with buildHelp.cfm

The buildHelp.cfm file holds the ColdFusion code needed to extract the right information from the database and present it (see Listing 2).

The page begins by defining default values for link colors. Next, a query is run against the specified data source to extract all information about the current page. If the help designer specified background or link colors, they will be used.

This page is the "hub" of the help system. It serves several functions, each of which is specified by a variable called "action." The allowable actions are:

- addNote
- updateDB
- moreHelp
- sendHelp
- showHelp

Each time this page is run, it checks to see which action is called for. The action specified determines which code is <cfinclude>d with the page. The code that called this page didn't indicate what action should be taken. The default action is "showHelp," and the showHelp.cfm page is <cfinclude>d.

FIGURE 5: More Help icon

Displaying Help

The showHelp.cfm page is quite concise (see Figure 4 and Listing 3).

Since all <cfinclude>d pages inherit the main page's variables, we can immediately begin by using the query "getPageInfo" that helpGlobals.cfm ran.

```
<!--- Query was already run from help
Globals.cfm --->
<cfoutput query="getPageInfo">
```

I mentioned earlier that one of the most popular features of the help system is the "user notes" function. This next bit of code creates a linked image to provide this.

```
<!--- Let users create their own notes.
This calls this same page and sends it the
url variable "action" set to "addNote"--->
<a
href="#cgi.script_name#?action=addNote&pageI
D=#url.pageID#"><img src="images/pencil.gif"
align="right" alt="Create User Note" bor-
der="0"></a>
```

Notice that the URL directs the browser to reload the current page – only this time with a different "action" called for. Now we're ready to display the help text and the users' notes that were returned by the query "getPageInfo." Finally, we provide a means for the user to ask for more help. The ambulance icon (see Figure 5) again calls the same page, sending the action "moreHelp" in the URL.

If you're beginning with ColdFusion, you may not be familiar with the idea of reusing a page by sending it different actions. It's worth studying this technique as it allows you to make your code far more readable and easier to maintain. In addition, the technique is central to writing code that complies with the new "Fusebox" standard (www.fusebox.org).

The following code pages should make clear how actions are linked to code.

The action "addNote" executes the code in Figure 6 and Listing 4.

This form sends the action "updateDB" to the main page. Here, the main page adds in the code in Listing 5.

Users who click on the More Help (ambulance) icon (see Figure 7) run the code in Listing 6.

The form in Listing 7 calls the page that sends e-mail off to the help folks specified in the application.cfm page.

Conclusion

If your users are like ours, you may be surprised by how many kudos you receive for your investment in creating this help system. Sadly, users are accustomed to being left on their own. A simple help system makes an enormous difference in their satisfaction ratings.

If you're interested in receiving a copy of the code and graphics for this help system, e-mail me at hal.helms@TeamAllaire.com.

About the Author

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

helms@TeamAllaire.com

Listing 1: helpGlobals.cfm

```
<!--- Parameterize this stuff to make it easy to change --->
<cfset IP = "http://127.0.0.1/tour/">
<cfset application.helpDSN = "helpSystem">

<!--- Code to spawn help window --->
<cfoutput>
<script language="JavaScript">
function showHelp(helpPage) {
 remote =
window.open("","remotewin","width=350,height=400,scrollbars=
yes");
 remote.location.href = "#IP#buildHelp.cfm?pageID=" + help-
```

Allaire
- ColdFusion Server 4.0.1 Enterprise for Windows
- ColdFusion Server 4.0.1 Enterprise for Solaris
- ColdFusion Studio 4.0.1
- ColdFusion Express 4.0
- JRun 2.3.3 for Windows
- JRun 2.3.3 for Solaris

Galileo Development System
- Intr@Vision Foundation

RadView Software
- WebLoad 3.5

RSW Software
- eTest Suite 3.5
- eTest Suite 3.6

SYS-CON Publications
- CFDJ Vol. 1 Issue:1
- CFDJ Vol. 1 Issue:2
- CFDJ Vol. 1 Issue:3
- CFDJ Vol. 1 Issue:4

ColdFusion Developer's Journal

Welcome to the ColdFusion Developer's Journal

Evaluation CD-ROM

This ColdFusion Developer's Journal CD-ROM was designed and distributed to give our readers an opportunity to evaluate some of the most innovative ColdFusion and Web development tools available. The participating companies supplied full-featured evaluation versions of their flagship products that will assist you in your Web development experience.

Come and visit our ColdFusionJournal.com Web Site!

Subscribe to ColdFusion Developer's Journal and save!

```
Page;
}
</script>
</cfoutput>

<!--- We don't want a Help icon on a Help page! --->
<cfif #GetFileFromPath(cgi.script_name)# NEQ "buildHelp.cfm">
 <!--- Check whether we have help info for this page --->
 <cfquery datasource="#application.helpDSN#" name="check-
ForHelp">
  SELECT pageID FROM Pages WHERE pageID =
'#GetFileFromPath(cgi.script_name)#'
 </cfquery>
 <!--- If not, show "No Help" icon --->
 <cfif #checkForHelp.RecordCount# EQ 0>
  <img src="images/noHelpIcon.gif"
border="0">
 <cfelse>
  <!--- If help is available created the linked icon --->
  <cfoutput>
  <a
href="javascript:showHelp('#GetFileFromPath(cgi.script_name)#
')">
   <img src="images/helpIcon.gif" width=15 height=15 bor-
der="0">
  </a>
  </cfoutput>
 </cfif>
</cfif>
```

Listing 2: buildHelp.cfm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<!---       Filename: buildHelp.cfm
        Programmer: hal.helms@utaweb.com
                  Date: 8/29/98
             Expects: url.pageID
 --->
<!---  Create default values for display --->
 <cfparam name="getPageInfo.textColor" default="black">
 <cfparam name="getPageInfo.linkColor" default="navy">
 <cfparam name="getPageInfo.vLinkColor" default="gray">
 <cfparam name="getPageInfo.aLinkColor" default="red">

<!---  Get help information for this page --->
<cfquery
 datasource="#application.helpDSN#"
   name="getPageInfo">
 SELECT * FROM Pages WHERE pageID = '#url.pageID#'
</cfquery>

<cfoutput query="getPageInfo">
<html>
<head>
 <title>#pageTitle#</title>
</head>

<!--- If Help specified background and link colors use them,
otherwise use defaults --->
<cfif #pageBGcolor# is not "">
 <body bgcolor="#pageBGcolor#" leftmargin="30" alink="#get-
PageInfo.aLinkColor#" link="#getPageInfo.linkColor#"
vlink="#getPageInfo.vLinkColor#">
<cfelseif #pageBackground# is not "">
 <body background="#pageBackground#" leftmargin="30"
alink="#getPageInfo.aLinkColor#" link="#getPageInfo.linkCol-
or#" vlink="#getPageInfo.vLinkColor#">
<cfelse>
 <body bgcolor="white" alink="#getPageInfo.aLinkColor#"
link="#getPageInfo.linkColor#" vlink="#getPageInfo.vLinkCol-
or#">
</cfif>
<p>
</cfoutput>
```

```
<!--- This is a multiuse page. The default action is
“showHelp”. If the user wants to add a note, url variable
action is “addNote”. AddNote then calls “updateDB”. Finally,
if user needs yet more help, s/he can click on the “More
Help” icon. The “moreHelp” action then calls the “sendHelp”
action.  Depending on the value of action, the rest of the
page will be included in this main page.--->
<cfparam name="url.action" default="showHelp">
<cfif #url.action# EQ "addNote">
 <cfinclude template="addNote.cfm">
<cfelseif #url.action# EQ "updateDB">
 <cfinclude template="updateDB.cfm">
<cfelseif #url.action# EQ "moreHelp">
 <cfinclude template="moreHelp.cfm">
<cfelseif #url.action# EQ "sendHelp">
 <cfinclude template="sendHelp.cfm">
<cfelse>
 <cfinclude template="showHelp.cfm">
</cfif>
</body>

</html>
```

Listing 3: showHelp.cfm

```
<!--- Query was already run from helpGlobals.cfm --->
<cfoutput query="getPageInfo">

<h2>#pageHeading#</h2>

<!--- Let users create their own notes. This calls this same
page and sends it the url variable "action" set to
"addNote"--->
<a
href="#cgi.script_name#?action=addNote&pageID=#url.pageID#"><
img src="images/pencil.gif" align="right" alt="Create User
Note" border="0"></a>

#helpText#
<br>
<hr align="center" noshade size="3"  width="70%">

<h3>User Notes</h3>
#userNotes#
<br>

<a href="#cgi.script_name#?pageID=#url.pageID#&action=moreHelp">
<img src="images/ambulance.gif" border="0" align="right">
</a>
</cfoutput>

<form>
<center><input type="button"  value="       OK       "
onClick="self.close()"></center>
</form>
```

Listing 4: addNote.cfm

```
<!--- Query was already run from helpGlobals.cfm --->
<cfoutput query="getPageInfo">

<h2>#pageHeading#</h2>

<!--- Return to Help screen--->
<a
href="#cgi.script_name#?action=showHelp&pageID=#url.pageID#">
<img src="images/info.gif" align="right" alt="Back to Help"
border="0"></a>

<h3>Add User Note</h3>

<form
action="#cgi.script_name#?pageID=#url.pageID#&action=updat-
eDB" method="post">
<textarea name="userNotes" cols="30" rows="3" wrap="PHYSI-
```

```
CAL"></textarea>
<p>
<div align="center"><input type="Submit" value="   Add Note
"></div>
</form>

</cfoutput>
```

Listing 5: updateDB.cfm

```
<!--- Add new note to the existing notes --->
<cfset oldString = "#getPageInfo.userNotes#">
<cfset newString = "#oldString##ParagraphFormat(form.user-
Notes)#<p>">

<!--- Update the database --->
<cfquery datasource="#application.helpDSN#" name="updateDB">
UPDATE Pages
Set userNotes = '#newString#'
WHERE pageID = '#url.pageID#'
</cfquery>

<!--- Now back to Help screen --->
<cfoutput>
<cflocation url="#cgi.script_name#?pageID=#url.pageID#">
</cfoutput>
```

Listing 6: moreHelp.cfm

```
<!--- Query was already run from helpGlobals.cfm --->
<cfoutput query="getPageInfo">

<h2>#pageHeading#</h2>

<!--- Return to Help screen--->
<a
href="#cgi.script_name#?action=showHelp&pageID=#url.pageID#">
<img src="images/info.gif" align="right" alt="Back to Help"
border="0"></a>

<h3>Request More Help</h3>
```

```
<form
action="#cgi.script_name#?pageID=#url.pageID#&action=send-
Help" method="post">
Please explain your concern, question, or comment:<br>
<textarea name="question" cols="30" rows="3" wrap="PHYSI-
CAL"></textarea>
<p>
<div align="center"><input type="Submit" value="   Send
Request   "></div>
</form>
</cfoutput>
```

Listing 7: sendHelp.cfm

```
<!--- Send email with page name and user question --->
<cfmail from="#application.name#" to="#emailList#"
server="ns1.utaweb.com" subject="Request for Help!"
type="html">
Submitted #DateFormat(now(), "m/d/yy")# at
#TimeFormat(now())#
<p>
<b>Application:</b> #application.name#
<p>
<b>Question/Comment:</b> #form.question#
</cfmail>

<!--- Now back to Help screen --->
<cfoutput>
<cflocation url="#cgi.script_name#?pageID=#url.pageID#">
</cfoutput>
```

CODE LISTING

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

Creating Subforms

Using inline frames, JavaScript and WDDX to represent a one-to-many relationship in an elegant manner

BY
MARK CYZYK

In the June issue of ***ColdFusion Developer's Journal*** (Vol. 1, issue 3), I hinted there might be a way to create subforms using inline frames. This article shows how – using a combination of inline frames, JavaScript, WDDX and, of course, CF.

Right now, inline frames are supported only by Internet Explorer, but because they're part of the official HTML specification it's hoped that the other main browser platforms will soon support them. Essentially, inline frames allow a developer to create a space on the screen that acts like a content island – a separate frame surrounded on all sides by the top frame. I'll show how this is done and the benefit of using inline frames instead of text areas to create a subform.

The other new technology used in this project, WDDX – the Web Dynamic Data Exchange – is an XML-based, Allaire-sponsored specification for transferring data over the Web between disparate programming environments. This project will use WDDX packets to transfer data between the JavaScript and ColdFusion environments as well as to transmit data from a form to an action template.

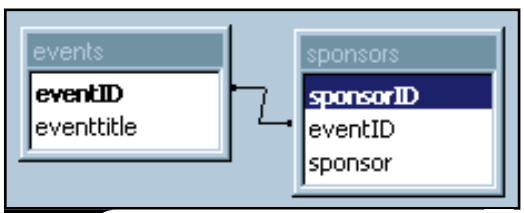


FIGURE 1: One-to-many relationship

Listing 1 represents the code for the input form for this project: a stripped-down, Web-based Events List. The code for the input form itself is on Lines 85–98. The form is simple, consisting of only a few elements. Two text boxes appear for the client to fill in – one for Event Title, the other for Event Sponsor. This data, however, exists in a one-to-many relationship in the back-end database, e.g., a single event can have multiple sponsors. This is the central problem that a subform is designed to resolve – how to represent a one-to-many relationship in an elegant manner on an input screen (see Figure 1).

Other form elements include an inline frame, a submit button and a hidden field that will contain a WDDX packet to be shipped off to the action template when the form is submitted.

Lines 92–94 create an inline frame on the screen. This frame, named "displayinlineframe", will be used to display the name of each Event Sponsor as it's entered. Because this element is named, it's exposed to JavaScript (much like the form itself); its values and what's displayed in it can therefore be manipulated via scripting.

Manipulating the values of the various form elements via JavaScript is the foundation underlying this project. The form works this way: the client enters a title in the Event Title text box (see Figure 2), then enters a Sponsor in the Event Sponsor(s) text box and clicks the >> button. The onClick() event of the >> button fires a JavaScript function that takes the data from the Event Sponsor(s) text box, pushes it onto a JavaScript array, then dumps the contents of that array to a WDDX packet that's stored as the value of the hidden "wddxpacket" form element. It then clears the contents of the inline frame and also dumps the contents of the updated JavaScript array there so the client can see that the Sponsor was added. When it's all done, the client clicks Submit and the contents of the form – including the WDDX packet containing the data of the JavaScript array intact – are transmitted to the action template for processing.

Let's analyze the JavaScripting on this form line by line.

Lines 5 and 6 simply import the WDDX and wddxDes packages that are included in the beta 2 version of the WDDX Software Development Kit available from www.wddx.org. These JavaScript packages allow for the serialization (or creation) of WDDX packets as well as the deserialization of a WDDX packet into native JavaScript format. Line 10 creates an instance of the WddxSerializer object that will be used further on in the script.

Lines 12 and 13 create some global variables, one of which ("eventSponsorArray") is an array that will be used to hold the value for each Sponsor as it's added.

Line 15 is where the fun begins. Lines 15–35 constitute the addIt() function, which is called whenever the >> button on the main form is clicked. The first thing this function does is to see whether the client has filled in the Event Sponsor text box ("eventsponsortextbox"). If the client is attempting to submit an empty box, an error alert pops up and processing of the addIt() function is aborted. Otherwise, processing proceeds.

Line 23 pushes the value of "eventsponsortextbox" onto the JavaScript array, using the value of the "counter" variable as the array element index. Line 24 then increments the counter in preparation for future values.

Lines 26 and 27, respectively, clear the display in the "eventsponsortextbox" and the inline frame.

Line 29 calls the buildFinalDisplay() function, which builds an HTML table that contains an item in each table cell from the JavaScript array. It then writes this table to the inline frame for display to the client.

Line 31 serializes the JavaScript array into a WDDX packet and writes it to the value of the "wddxpacket" hidden form field.

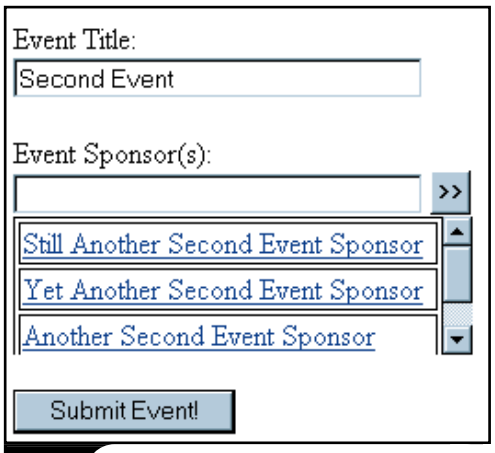


FIGURE 2: The insert form

	eventID	eventtitle
▶	7	First Event
	8	Second Event
	9	Third Event
*	(AutoNumber)	

FIGURE 3: The Event table

Line 33 simply returns focus to the now empty "eventsponsortextbox" in preparation for input of the next Event Sponsor.

As items are entered into the Event Sponsor text box and submitted via the >> button, they're pushed onto a JavaScript array. A WDDX packet is secretly created and stored in a hidden form field containing this data, ready to be posted to the action template whenever the client clicks the form's Submit button.

A question that might arise: Why an inline frame? Why couldn't a regular text area be used for this project? In fact, a text area *can* be used for this project and works fine for what's outlined so far. But one thing that can't be done within the confines of a text area is to make its contents interactive – HTML and JavaScript coding within a text area will be rendered as straight text, not as an interactive Web document. But why is it important to be able to do that anyway?

Suppose the client entered an Event Sponsor and clicked the >> button. The client then noticed that there's either an error in his or her entry or the entry isn't really valid, i.e., it should either be updated or deleted altogether. Using a text area, there's no way to interact further with the data, and the client would be stuck with what was originally input. An inline frame, however, creates a space for a wholly separate HTML document to appear within the confines of its parent document. This is accomplished by writing a combination of HTML and JavaScript to wrap each element of the HTML table generated by the addIt() function; Line 41 of the addIt() function not only dumps the array element to an HTML table cell, it also wraps that element in a hypertext link that, when clicked, calls the deleteFromArray() function back on the parent document.

This function allows the client the opportunity to either update or delete a previously submitted item by setting the value of the "eventsponsortextbox" to the value of the element needing updating or deleting. It then moves the focus to that text box. This occurs on Lines 53–54.

Next, the item is deleted from the JavaScript array on Line 55. But because of the way JavaScript arrays function, only the value of the item is deleted from the array, not the array element itself, which remains intact with its original element index – only now its value is "undefined." This annoying feature must now be dealt

with – we certainly don't want multitudes of undefined array elements cluttering up our data structures! One way to deal with the situation is to create an alternate, temporary array and push all undefined items in the original array onto it. The original array can then be reset and repopulated with clean data from the temporary array. This is done on Lines 56–65. Once done, the array can be dumped to the inline frame and to the hidden "wddxpacket" form field, just as it is in the addIt() function. By deleting, then resubmitting, items from the original array, the client can update or delete elements before posting the data to the action template.

Another feature of the JavaScripting on this form should be pointed out before we take a look at the action template. The body tag of the HTML document contains an onLoad() event handler, which calls the initialize() JavaScript function each time the page is loaded. This function checks whether a value has already been declared in the hidden "wddxpacket" field. This is done to solve a curious problem that crops up when, for instance, the client leaves the page and returns, which might occur if there's an error condition on the action template and the client returns to the form to resubmit. When the client leaves the form, all JavaScript variables -- including the central JavaScript array -- are deleted. However, the values of the form fields are cached. If the client returns to the form, it will appear that everything remained the same. The values in the inline frame were cached, so they'll remain intact, as will the contents of the "wddxpacket" field. But if the client then decides to add, update or delete anything, problems arise. The original JavaScript array was flushed when the client left the page, so what appears on the screen in no way reflects the contents of this crucial data structure (which at this point is empty). The initialize() function remedies the situation by first checking for the existence of the WDDX packet cached in the "wddxpacket" form field.

	sponsorID	eventID	sponsor
▶	11	7	First Event Sponsor
	12	7	Another First Event Sponsor
	13	8	Second Event Sponsor
	14	8	Another Second Event Sponsor
	15	8	Yet Another Second Event Sponsor
	16	8	Still Another Second Event Sponsor
	17	9	Third Event Sponsor
	18	9	Another Third Event Sponsor
	19	9	Yet Another Third Event Sponsor
*	(AutoNumber)	0	

FIGURE 4: The Sponsor table

If it finds this packet, it deserializes it into a native JavaScript array, restoring what was flushed when the client left the page. Everything is back to normal – problem solved.

Once the client completes the form and the Event Title and "wddxpacket" form fields are properly populated, the form is submitted to the action template for processing. Listing 2 represents the code for this action template.

The first thing the action template does is wrap all interactions with the database in <CFTRANSACTION> tags to preserve data and referential integrity. It then (Lines 9–12) inserts the contents of the "eventtitle" form field into the Events table. Remember, each singular event can have multiple Event Sponsors, so the next thing to do is to determine the key of the event just entered into the Events table. It'll be used as the foreign key for each record subsequently entered into the Sponsor table (Lines 14–17).

Lines 19–23 extract the data from the "wddxpacket" form field and deserialize them into a native ColdFusion array. This is the beauty of WDDX – it enables the developer to transmit complex data structures from one programming platform to another. In this case we translated a JavaScript array on the previous form into a WDDX packet, transferred that packet to the action template, then deserialized it into a ColdFusion array. Pretty nifty!

Once the ColdFusion array, populated by multiple Event Sponsors, is properly in place, it can be looped through. It inserts each element into the Sponsors table using the Event ID as the foreign key linking back to the Events table. Figure 3 illustrates the state of the Events table once this is accomplished; likewise, Figure 4 represents the state of the Sponsor table. Notice the mapping of the foreign keys from the Sponsor table back to their corresponding records in the Events table.

Once that's done, the <CFTRANSACTION> tag is closed and the process is complete.

Why create subforms this way? Why use JavaScript and WDDX to do this when, as detailed in ***CFDJ***'s June issue, a developer can create subforms using the <CFGRID> tag? Although not illustrated in this article, the technique outlined above allows the developer to control the client's domain of choices by using a data-driven SELECT box instead of a free-form text box for data input. This isn't possible using the <CFGRID> tag (but would be a welcome addition!). It also, in many ways, is a more intuitive way to arrange information on the screen and thus is easier to use from the client perspective.

By using a combination of ColdFusion, JavaScript, inline frames and Allaire's WDDX format for transferring data, the developer can create a subform that's screen-efficient and elegant to use. And because all processing is done client-side, subforms are generated and rendered exceedingly fast. This, at least, is an instance where pushing application code out to the client for processing makes for a more efficient and effective browsing experience.

ABOUT THE AUTHOR

Mark Cyzyk is a Web developer for Johns Hopkins University and Nine Web, LLC, where he creates dynamic, data-driven Web applications using ColdFusion, JavaScript, Cascading Stylesheets and advanced HTML.

MCYZYK@JHU@EDU

Listing 1: The Input Form

```
1 <html>
2 <head>
3 <title>Events List</title>
4
5 <script src="/cfide/scripts/wddx_sdk/wddx.js"
  language="javascript"></script>
6 <script src="/cfide/scripts/wddx_sdk/wddxDes.js"
  language="javascript"></script>
7
8 <script language="javascript">
9
10 mySerializer = new WddxSerializer;
11
12 var eventSponsorArray = new Array();
13 var counter = 0;
14
15 function addIt() {
16
17 if (document.form1.eventsponsortextbox.value == "") {
18       document.form1.eventsponsortextbox.focus();
19       alert("You must include a value in the text box!");
20       return false;
21 }
22
23 eventSponsorArray[counter] =
   document.form1.eventsponsortextbox.value;
24 counter++;
25
26 document.form1.eventsponsortextbox.value = "";
27 frames.displayinlineframe.document.open();
28
29 buildFinalDisplay();
30
31 serializeIt();
32
33 document.form1.eventsponsortextbox.focus();
34 return true;
35 }
36
37 function buildFinalDisplay() {
38 var finaldisplay = "";
39 finaldisplay = "<table width=100% border=1
   bordercolor=black>"
40 for(var i = counter - 1; i >= 0; i--){
41 finaldisplay = finaldisplay + "<tr><td><a href=
   javascript:top.deleteFromArray(" + i + ")>" + 42
   eventSponsorArray[i] + "</a></td></tr>";
43 }
44 finaldisplay = finaldisplay + "</table>";
45 frames.displayinlineframe.document.write(finaldisplay);
46 }
47
48 function serializeIt() {
49 myArrayInWDDX = mySerializer.serialize(eventSponsorArray);
50 document.form1.wddxpacket.value = myArrayInWDDX;
51 }
52
53 function deleteFromArray(item) {
54 document.form1.eventsponsortextbox.value =
   eventSponsorArray[item];
55 document.form1.eventsponsortextbox.focus();
56 delete(eventSponsorArray[item]);
57 tempArray = new Array();
58 counter = 0;
59 for (var i = 0; i < eventSponsorArray.length; i++) {
60       if (eventSponsorArray[i] != null) {
61                tempArray[counter] = eventSponsorArray[i];
62                counter++;
63       }
64 }
65 eventSponsorArray = new Array();
66 eventSponsorArray = tempArray;
67 frames.displayinlineframe.document.open();
68 buildFinalDisplay();
69 serializeIt();
```

```
70 }
71
72 function initialize() {
73 if (document.form1.wddxpacket.value != "") {
74       MyArrayDeserializer = new WddxDeserializer;
75       eventSponsorArray = MyArrayDeserializer.deserialize
         (document.form1.wddxpacket.value);
76       counter = eventSponsorArray.length;
77       buildFinalDisplay();
78 }
79 }
80
81 </script>
82
83 </head>
84 <body onLoad="initialize()">
85
86 <form method="post" action="action.cfm" name="form1">
87 Event Title:<br>
88 <input type="text" size=30 name="eventtitle">
89 <p>
90 Event Sponsor(s):<br>
91 <input type="text" size=30 name="eventsponsortextbox">
92 <input type="button" name="eventsponsorbutton"
   onClick="addIt()" value=">>"><br>
93 <iframe name="displayinlineframe" width=250 height=75
   marginwidth=0 marginheight=0 frameborder=0>
94 Your browser does not support inline frames.
95 </iframe>
96 <p>
97 <input type="hidden" name="wddxpacket" value="">
98 <input type="submit" value="Submit Event!">
99 </form>
100
101 </body>
102 </html>
```

Listing 2: The Action Template

```
1 <html>
2 <head>
3 <title>Events List Action</title>
4 </head>
5 <body>
6
7 <cftransaction>
8
9 <cfquery datasource="subforms" name="insertEvent">
10 INSERT INTO EVENTS(EVENTTITLE)
11 VALUES('#FORM.EVENTTITLE#')
12 </cfquery>
13
14 <cfquery datasource="subforms" name="getID">
15 SELECT         MAX(EVENTID) AS theID
16 FROM  EVENTS
17 </cfquery>
18
19 <cfwddx
20 INPUT = "#FORM.WDDXPACKET#"
21 OUTPUT = "thisArray"
22 ACTION = "WDDX2CFML"
23 >
24
25 <cfloop from="1" to="#ArrayLen(thisArray)#" index="counter">
26 <cfquery datasource="subforms" name="insertSponsors">
27 INSERT INTO SPONSORS(SPONSOR, EVENTID)
28 VALUES('#thisArray[counter]#', #getID.theID#)
29 </cfquery>
30 </cfloop>
31
32 </cftransaction>
33
34 <b>Done!</b>
35
36 </body>
37 </html>
```

CODE LISTING

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

Architecting Enterprise-Level ColdFusion Applications Part 1

One step closer to emulating the world

BY ED DONOHUE & BENJAMIN ELMORE

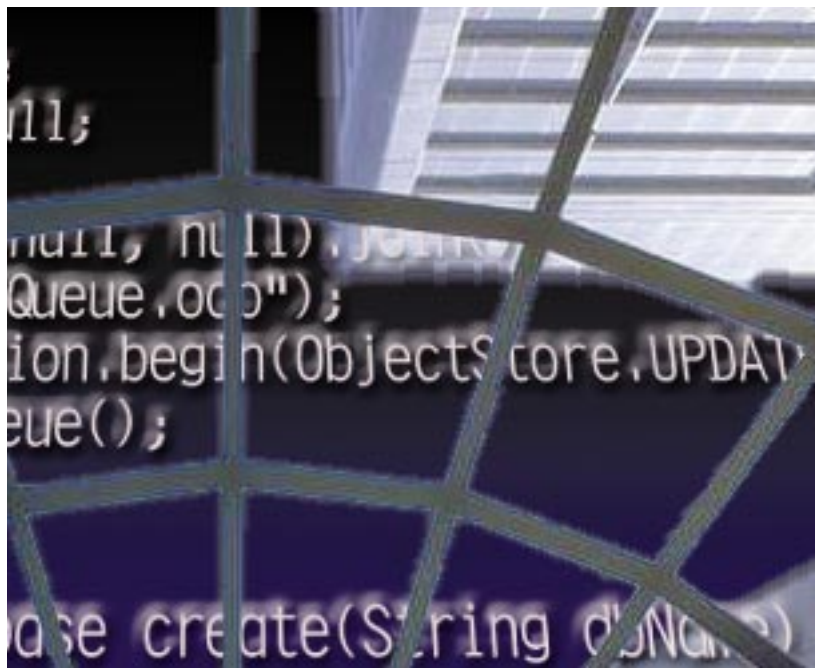
Welcome to the first of several articles that focuses on the approach rather than the coding of ColdFusion applications.

Before you flip to the next article, we encourage you to read on – this topic doesn't have to be dull…instead, it's our intent to help serious ColdFusion developers, like yourself, become more effective at building world-class Web-based applications.

We'll discuss several approaches pioneered in the client/server world that have driven the industry toward component-based architecture designs, and look at how these techniques will benefit ColdFusion developers responsible for building robust, enterprise-level Web applications.

Many developers are now in the midst of transitioning themselves from a client/server mind-set to a browser/server one. There is a definite need at this stage to start thinking in terms of components before getting too deep into building enterprise-level Web applications. Our multipart series will address this by illustrating concepts as seen through the eyes of an average ColdFusion developer who is making this change.

We will also demonstrate ways to leverage design patterns, universal interfaces and distributed components that can immediately be applied to current Web projects.



Transitioning to *n*-Tier

Many of us don't really understand what a multitier environment is or what the true definition of a "tier" is. *Webster's Dictionary* defines a tier as a "row, rank, or layer." Therefore, if we have a two-tier application, then we can safely say that it comprises two layers (see Figure 1).

FIGURE 1: Two-tier environment

Most of us are at a stage in our career where we're just getting comfortable with two-tier client/server applications using such programming languages as VB, Delphi, FoxPro, PowerBuilder, etc., which connect to a database.

This is pretty much the world client/server developers live in. In fact, even those of us who are more familiar with creating static HTML Web pages can be considered two-tier developers since the connection between the browser and the Web server is essentially two tiers. It's when we start looking at server-side programming that things begin to transition from a two-tier to three, four, five…*n*-tier environments.

Let's consider the landscape of a typical ColdFusion application (see Figure 2). We'll discover that this environment comprises four tiers or layers. The browser has a connection to the Web server, the Web server has a connection to the ColdFusion application server and the ColdFusion application server has a connection to a database. We can think of this as a four-tier environment.

However, we really should think of this environment as a collection of three client/server environments. The browser is a client to the Web server, the Web server is a client to the ColdFusion server and the ColdFusion server is a client to the database server. As you can see, our client/server expertise will transfer nicely over to the *n*-tier environment.

Throw any number of additional servers into this model and we can still break it down to a collection of client/server connections. Visualizing an *n*-tier environment this way will help remove the complexity and allow us to focus on just two tiers at a time.

Building Upon Past Successes (Best Practices)

Innovations in software development have been with us since the days of the punch card. We've all learned a great deal from the experiences of others in developing software. Each new generation of programming languages was another step in the evolution of application development. Procedural languages such as COBOL, FORTRAN, PL/1, etc., were some of the first to promote reuse by providing a way for developers to code procedures or functions, and to include prewritten code as well into their software applications.

As the evolution progressed toward 4GL, a shift away from procedural programs gave way to a new and exciting event-driven model. The use of object-oriented programming (OOP) features – such as inheritance, encapsulation and polymorphism – in our applications allowed us to start a new chapter on how software systems can be constructed better. We discovered how to create objects that were self-contained and reusable. These objects were also capable of having organic traits such as behavior, reproduction and life expectancy. But the story doesn't end here. OOP is just another step in the evolutionary path of software development. Our journey through this evolution (see Figure 3) is now focused on components and distributed systems, and it's our challenge to pave the way for the next generation.

Components and Distributed Systems

Each evolutionary step we take on the software development path brings us one step closer to emulating the world around us. OOP has allowed us to create classes that mirror real-world objects. Having these classes work in a reusable unit (referred to as a component) marked the beginning of the transition toward Distributed and Component Based Systems (DCBS). DCBS requires the understanding and usage of components, distributed architecting through partitioning and integration at the system level.

Before we discuss these three concepts, let's consider the advantages of developing components and look at what the differences are between OOP and DCBS.

A component is considered "a constituent part; an ingredient; a collection of objects." Components are all around us. Let's use the simple light bulb as an example.

First of all, light bulbs obviously come in all shapes and sizes to fit the requirement of the host system. Each one is made from the same basic materials but may be manufactured with varying types of metal connection bases that we will refer to as the "public interface." Like any well-made component, light bulbs are made up of several different objects working together (filament, glass shell, metal base, etc.). Consumers can decide on how many watts the light bulb should have as well as its color, shape and size.

The key to a component's portability and usability is in its public interface. Well-designed public interfaces allow components to be "snapped" together to create more useful and complex components. Moreover, other advantages emerge from creating components that extend well beyond the flexibility of being able to exchange one type of light bulb for another or the ability to use a standard light bulb design for different types of lighting systems. Components can be easily and inexpensively manufactured in great numbers using an assembly-line approach. The assembly-line approach is precisely what we're looking to achieve with our software development efforts. Mass production allows us to break up the work so that we can have our more advanced developers engineer components and let the less experienced developers work on assembling them. This maximizes the varied resources found in most IS shops. If more IS managers looked at their departments from a manufacturing point of view, they would be surprised at how many ways they could improve efficiency and productivity.

FIGURE 2 Four-tier environment

Design for components has its origins in OOP. Developers are able to leverage the reusability of these components along with enjoying minimized maintenance cycles. The difference between DCBS and OOP is that OOP is restricted to the existing application. DCBSs are designed to be ubiquitous.

A recent trend in OOP is application partitioning. Everyone has heard "encapsulate your business logic" or "don't code logic in the UI." While this is considered good design standards, in DCBS it's a must. The partitioning

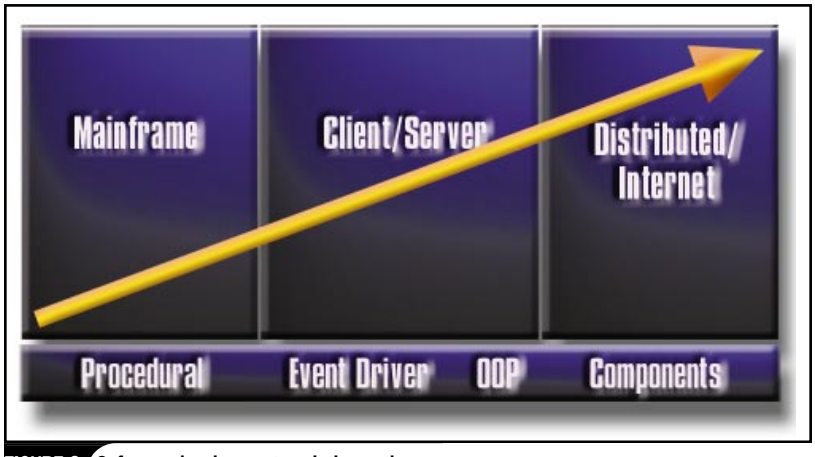


FIGURE 3 Software development evolution path

ABOUT THE AUTHORS

Ed Donohue, director of Internet technologies at CGI Information Systems Consulting Services, Inc., is a national speaker and Internet evangelist with over 16 years' experience designing and developing applications across a wide variety of industries.

Benjamin Elmore, a consultant at CGI Information Systems Consulting Services, Inc., is a Certified Allaire Instructor and a Certified Powersoft Developer Associate.

model is broken up into three sections (see Figure 4): "User Interface" (windows, Web page), "System Management" (transaction to database, management of user info) and "Problem Domain" (business-specific logic, service classes).

Today, most applications are deployed to a single platform, either Web or PC based. New DCBS systems will allow us to utilize the "Problem Domain" partition, and snap in multiple-user interfaces in front of them. Centralizing your business logic allows for easier maintainability of the overall applications, and reusability not only across multiple front ends but also across applications.

The last concept we're going to discuss is integration. As we learn how to properly design and implement distributable components in our applications, we'll be building a repository of reusable business-specific logic. This is very similar to the inventory bins used at manufacturing plants. Integration of these components is the hallmark of DCBS. DCBS isn't limited to a specific language. It provides a way for us to integrate "Best of Breed" technologies into our software development efforts. Whether the development language is Java, C++ or ColdFusion, DCBS allows us to use the best tool for the job and provides the ability to assemble prebuilt/pretested components into a completed system.

FIGURE 4 Partitioning model

Intercomponent Communication

ColdFusion contains many features that help us solve new challenges that arise when developing Web enabled/component applications. For example, ColdFusion helps us manage inherent problems such as statelessness. But what can we do to simplify the way disparate components communicate? Later in our series, we'll look at creating universal interfaces using XML and ColdFusion's WDDX technology.

Summary

There are only two ways to build systems -- hack them together or architect them. The tree-forts we built as kids were prime examples of how to hack something together. Would you have built the house you live in today using that same approach? Of course not, but we're all guilty of hacking software systems together.

Applying architecture to our applications doesn't mean spending weeks inside a word processor. We simply need to put a little thought up front to developing a plan and learn to apply basic design principles in order to achieve it.

Over the next few articles we'll be spending time elaborating on architecture and the approaches and techniques that make it work.

BENJAMIN.ELMORE@CGIUSA.COM
ED@EDONOHUE.COM

Macromedia Generator

The sky's the limit with Generator

BY ANDREW STOPFORD

Shockwave Flash is all over the Web. Its eye-catching, interactive animations have found a place on today's Web, and most Web firms use both Flash and its bigger brother on both their own and their clients' Web sites.

Just over a year and a half ago, Macromedia introduced a product called Generator, and Flash got even better.

What Is Generator?

Generator allows Flash to dynamically change its own content, be it text or colors. It can even insert graphics, change the position of elements in the movie, and query databases for scrolling tickers and graphs. The key word to Generator is dynamic.

How Can I Use Generator?

Generator is split into two parts. Offline is a command-line-driven program (e.g., DOS) where for a given Generator file (or Shockwave Template – SWT) it will create an SWF (Shockwave Flash) file. Offline Generator can also create images, server-side and client-side image maps, and projector files (for running as an EXE file). Sample use would be:

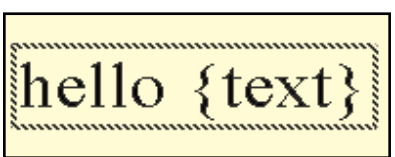


FIGURE 1

FIGURE 2

```
Generate -swt myfile.swt myfile.swf
```

The command Generate-help gives a full syntax of the Offline Generator and it's explained in the Generator documentation.

Online Generator is installed as a Java servlet to your Web server, running under JRun. This allows the Web server to process a request for an SWT. The servlet processes the SWT file, adding data and changing the contents of the file as indicated within it. What the servlet returns is a finished SWF ready to be run within the Web page. Online Generator can also create images as well as SWF files.

The process is quite transparent, and the user won't know that the Web server has just built the Shockwave animation they see in their browser.

Creating an SWT File

I won't go into in-depth use of the Macromedia Flash product here. As part of an online demo I will show how you can use it with Generator. Open up Flash and add a text item as shown in Figure 1.

Note the {} brackets. They indicate a Generator variable. Whatever data we now pass to the SWT file, if it points to that variable, then that variable will get that value. Make sure you save the file as an SWT file.

Using ColdFusion with Generator

I will demonstrate here how to use both the offline and online Generator with ColdFusion. As described, the Online Generator is a Web server component for creating Shockwave content dynamically. This allows us to pass to the SWT file parameters for any variables that we set in the movie.

First I created our HTML page.

```
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF">
<form method="post" action="out.cfm" name="">
  <table width="75%" border="0">
    <tr bordercolor="#FFFFFF" bgcolor="#CCFFFF">
      <td>Enter your name:
        <input type="text" name="text">
        <input type="submit" name="Submit" value="OK">
      </td>
    </tr>
  </table>
  </form>
</body>
</html>
```

This is a simple HTML page for sending some text to our SWT file. Next I created our CFM page to process that data and pass it to the SWT file.

```
<cfparam name="text" default="world">

<cfset text=URLEncodedFormat(text)>

<html>
<head>
<title>Generator CF</title>
<meta http-equiv="Content-Type"
```

```
content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF">
<div align="center"><object
classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#3,0,0,0" width="200"
height="200">
    <param name="SRC"
value="mov.swt?text=<cfoutput>#text#</cfoutput>">
    <embed src="mov.swt?text=<cfoutput>#text#</cfoutput>"
pluginspage="http://www.macromedia.com/shockwave/download/" type="application/x-shockwave-flash" width="200"
height="200">
    </embed>
  </object> </div>
</body>
</html>
```

It's worth noting the way in which the data is passed to the SWT file.

```
mov.swt?text=<cfoutput>#text#</cfoutput>
```

Note that we called our generator variable "text" (as shown in Figure 1), and here we pass it the value taken from our HTML page. The end result is that our Generator variable takes the value of the HTML page, as shown in Figure 2.

Offline Use

Since Offline Generator is a command-line-driven program, how can we use it from within our Web pages? Well, you're faced with two choices: use a CFM tag for executing command-line programs or use the CFM_Generate tag.

The aim is to execute the Offline Generator, and the CFM_Generate tag makes this a bit easier to do. The tag comes precompiled with the ColdFusion Generator SDK that is freely available to download from the Macromedia Web site (www.macromedia.com/software/generator). Once installed, it allows you to execute the Offline Generator from within your ColdFusion pages. Full instructions do come with the SDK, but to demonstrate, here is sample code.

```
<CFX_Generate
      TEMPLATE="c:\webroot\myfile.swt"
      TYPE="swf"
       FILE="c:\webroot\myfile.swf">
```

This is the same as using:

```
Generate -swt myfile.swt myfile.swf
```

at the command line.

Note that the path to the Offline Generator must be in your system path. With this tag you can also create image maps and set options to write to Generator's log files.

Summary

I have shown some simple uses of Generator and how you can start to use your ColdFusion code with it. The sky really is the limit with Generator, and the results can be astounding.

ABOUT THE AUTHOR

Andrew Stopford, a freelance Web developer from south Manchester in the UK., has lent his hand to many Generator sites around the world. He's kept busy answering questions that the Generator community posts in the Macromedia Generator NG and other information sites. He's also the creator of "Kimmuli," a code tool for Generator.

A.STOPFORD@LEXON.CO.UK

ADVERTISING INDEX

ADVERTISER	URL	PH	PG
ABLE SOLUTIONS	WWW.ABLECOMMERCE.COM	360.253.4142	2
ALIVE.COM	WWW.ALIVE.COM	206.674.7700	17
ALLAIRE	WWW.ALLAIRE.COM	888.939.2545	31,45,47,55
BACKSOFT	WWW.BACKSOFT.COM	888.222.6047	15
BIZNIZ WEB	WWW.WEBPUBLISHINGTOOLS.COM	281.367.4016	58
CATOUZER	WWW.CATOUZER.COM	604.662.7551	67
CFDEV.COM	WWW.CFDEV.COM	800.854.7838	54
COMPUTERJOBS.COM	WWW.COMPUTERJOBS.COM		7
CONCEPTWARE AG	WWW.CONCEPTWARE.COM	061.964.7320	4
CYSCAPE	WWW.CYSCAPE.COM	800.932.6869	41
DATARETURN	WWW.DATARETURN.COM	800.767.1514	13
FINDADEVELOPER.COM	WWW.FINDADEVELOPER.COM	440.257.6690	58
FINDAHOST.COM	WWW.FINDAHOST.COM	440.257.6690	44
FUNDERE SOFTWARE	WWW.FUNDERE.COM	310.392.1005	19
FUSION FX	WWW.FUSIONFX.COM	800.780.2422	36
GALILEO	WWW.GALILEODEV.COM	770.643.9176	9, 61
HOSTPRO	WWW.HOSTPRO.NET	888.638.5831	37
INFOBOARD	WWW.INFOBOARD.COM	800.514.2297	54, 59
INFORUM SOLUTIONS	WWW.INFORUMSOLUTIONS.COM	978.887.0080	21
INTERMEDIA	WWW.INTERMEDIA.NET	650.424.9935	68
MACROMEDIA	WWW.MACROMEDIA.COM	800.457.1774	3
NEO TECH	WWW.CF-TRAINING.COM	973.540.9672	61
OMNIKRON SYSTEMS	WWW.OMNIKRON.COM	818.223.4126	54
ON-LINE DATA SOLUTIONS	WWW.COOLFUSION.COM	516.737.4668	57
ORIGENT	WWW.ORIGENT.COM	617.623.8044	49
PIRANHANET	WWW.PIRANHANET.COM	206.374.0880	27
RADVIEW SOFTWARE	WWW.RADVIEW.COM	888.RADVIEW	51
RSW SOFTWARE	WWW.RSWSOFTWARE.COM	508.435.8000	26
SITEHOSTING.NET	WWW.SITEHOSTING.NET	888.463.6168	42
SKY TECHNOLOGIES	WWW.SKYTECHNOLOGIES.COM	201.883.9373	43
VIRTUALSCAPE	WWW.VIRTUALSCAPE.COM	212.460.8406	48
WORLDWIDE NETFAST	WWW.NETFAST.NET	800.362.9004	57

Able Solutions

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

11700 NE 95th Street, Suite 100, Vancouver, WA

www.ablecommerce.com • 360 253-4142

Alive.com

You don't have to be an HTML programmer to create media-rich e-shows for your Web site. Use Alive's templates and prompts to quickly create your own slides, add audio, video and graphics, then synchronize it all as easily as clicking your mouse button. Need to put it on the intranet? Or the Web site? Yesler Alive could well be the most time and cost effective communication tool you'll ever own.

83 S. King St., Ste 414, Seattle, WA 98104

www.alive.com • 888 386-9969

Backsoft Corp.

BackTalk for SAP Web application server is the fastest way to build and deliver scalable applications that integrate browser, server and SAP database technologies. BackTalk for SAP is the first web-enabled solution for the SAP R/3 system which uses Allaire's ColdFusion Technology. It leverages the scalability of Allaire's ColdFusion Architecture and builds an intuitive development tool to Web-enable SAP.

5971 Cattleridge Blvd, Suite 101, Sarasota, FL 34232

www.backsoft.com• 941 378-2325

Catouzer, Inc.

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

1228 Hamilton Street, Suite 501, Vancouver, B.C. V6B 2S8, Canada

www.catouzer.com • 604 662-7551

CFDEV.com

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

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

Computerjobs.com

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

3200 Windy Hill Road, Suite 700 West, Atlanta, GA 30339

www.computerjobs.com

Conceptware

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

Industriestraße 30-34, Eschborn, Germany 65760

www.conceptware.com +49-(0)6196-47 32-0

Data Return Corporation

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

801 Stadium Dr., Ste 117, Arlington, TX 76011

www.datareturn.com • 800 767-1514

Fundere Software

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

18 Horizon Avenue,. Venice, CA 90291

www.fundere.com 310-392-7805

Fusion FX, Inc.

Fusion FX, Inc., is a full-service Web site design and hosting company. Our strength in ColdFusion Development, as well as with custom AbleCommerce Store building, and the fact that we have a Team Allaire member on staff will give you peace of mind because you know your ColdFusion development is in good hands.

819 Peacock Plaza, Suite 535, Key West, FL 33040
www.fusionfx.com • 800 780-2422

Galileo Development Systems

The Intr@Vision family of products provides the basis for automating your business processes over the web using a standard, extensible framework. With a focus on reducing the cost of doing business and improving cash flow, Intr@Vision Time and Intr@Vision Expense offer a web-based alternative to your current paper processes. Contact us today for more information on our products.

2695 Long Lake Dr., Roswell GA 30075
www.galileodev.com • 770 643-9176

HostPro

HostPro offers packages to suit just about anyone's growing bandwidth, storage and feature requirements. Our web site hosting solutions are packaged to meet specific needs based on popular criteria. This includes UNIX and NT packages, e-commerce solutions that range from simple to turn-key, databases like mSQL to mission-critical MS SQL, multimedia like RealAudio, RealVideo and all the programs and languages to extend your site's capabilities

3250 Wilshire Blvd., Suite 707, Los Angeles, CA 90010
www.hostpro.net • 213 252-9779

INFORUM Solutions

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

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

Intermedia, Inc.

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

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

Macromedia Corp.

Macromedia is the leading developer of cross-platform software tools for digital media creation and publishing. Macromedia's products are used by organizations to create and deliver interactive applications that use a full range of media, from text and graphics, to animation. The product line includes Director, Authorware, ShockWave, FreeHand, and Dreamweaver.

600 Townsend Street, San Francisco, CA 94103
www.macromedia.com • 415 252-2000

NetFast

NetFast strives to become the foremost leader in Web application software. Our applications empower and enable function-building efforts of Web developers and designers. These ready-to-use software components can be bundled into existing Intranet and Internet sites to increase usability, drive traffic and differentiate online offerings. The fusion of application development and hosting provides substantial revenue opportunities for VARs and integrators of NetFast products.

6699 Port West Drive, Suite. 130, Houston, TX 77024
www.netfast.net 1-800-362-9004

Omnikron Systems

Omnikron Systems, Inc., has grown into a world-class consulting firm that provides not only unmatched services, but also a set of values and commitments that are rarely seen in today's competitive environment. With a focus that continues to be on absolute customer satisfaction, we strive to deliver the finest solutions that are on time, cost effective and exceed your expectations.

5016 North Parkway Calabasas, Suite 101, Calabasas, CA 91302
www.omnikron.com 818.591.7890

On-Line Data Solutions

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

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

PiranhaNet

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

2112 6th Avenue, .Seattle, WA 98121
www.piranhanet.com 206.374.0880

RadView Software Inc.

RadView Software, mkers of WebLoad, specializes in developing Web application testing solutions. Our mission is to deliver best-of-breed scalability and integrity testing tools developed specifically to meet the evolving needs of Web developers, testers and IT resources. Our vision is to provide high-performance, easy-to-use and cost-effective testing solutions to a broad market.

1050 Waltham Street, .Lexington, MA 02421
www.radview.com 1-888-RADVIEW

RSW Software

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

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

Sky Technologies

SKY Technologies is a leading systems integration firm specializing in high-end, reliable information technology solutions for businesses of all sizes. We identify the best-of-breed products through our continuous industry research., and partner with leading hardware and software developers. Adding our high-quality support and service means we deploy scalable, results-oriented IT solutions tailored to our clients' needs.

411 Hackensack Avenue, Hackensack, NJ 07601-6406
www.skytechnologies.com 201-883-9373

Sitehosting.NET

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

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

Virtualscape

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

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

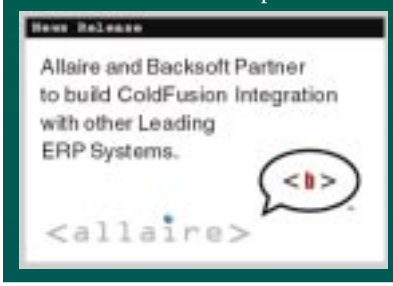
To place an ad in the ColdFusion Marketplace contact Robyn Forma at 914 735-0300

COLDFUSION MARKETPLACE

CFDJnews

Allaire and Backsoft to Integrate SAP R/3 and ColdFusion

(Cambridge, MA & Sarasota, FL) – Allaire Corporation and Backsoft Corporation have announced a strategic partnership that will allow companies to Web-enable their SAP R/3 applications. Backsoft is bringing to market the <b> Talk EAI server, which connects the ColdFusion Web application server with leading ERP systems, enabling the rapid development of scalable enterprise and e-commerce Web solutions.



Using ColdFusion 4.0 and <b>Talk 2.0, SAP R/3 customers will be able to quickly and easily extend their SAP applications to the Web. As a result, they will be able to build new browser-based business automation systems on the SAP R/3 platform, add new functionality to existing SAP applications and deliver e-commerce solutions that are fully integrated with an SAP R/3 backend.
www.allaire.com / www.Backsoft.com

CareerBuilder and ComputerJobs.com Partner to Assist IT Job Seekers

(Reston, VA) – CareerBuilder, Inc., and ComputerJobs.com have teamed together to make it easier for IT job seekers and employers to find each other. Through the partnership, IT professionals can access more than 2 million job postings. At the same time, employers can select computerJobs.com as a posting destination from the CareerBuilder Network.
www.careerbuilder.com / www.computerjobs.com

Conceptware a Sponsor of Allaire Developer Conference

(Eschborn, Germany) – Strengthening its partnership with Allaire, Conceptware will participate as a sponsor in the First Annual World-wide Allaire Developer Conference at the Sheraton Boston Hotel and Towers from October 24 to 26.

Conceptware will present a professional tool, GateBuilder, which was developed on the basis of the leading Web development tool, to a wide range of people at the conference.
www.conceptware.com

Omnikron Selected as Allaire Premium Partner

(Calabasas, CA) – Omnikron Systems Inc. has partnered with Allaire Inc. to sell and develop Allaire ColdFusion solutions and provide custom Web application development in southern California.
www.omnikron.com

Introducing Allaire Spectra

(San Francisco, CA) – Allaire Corporation has launched Allaire Spectra, a packaged system for next-generation content management, e-commerce and personalization. Other features and capabilities include workflow and process automation, roles-based security, business intelligence and syndication.
www.allaire.com

Ableauctions Selects Allaire to Launch Its E-Commerce Presence

(Cambridge, MA) – Ableauctions.com, a leading auctioneer of office equipment, computers, furniture and industrial equipment, will launch its online presence with an e-commerce system built and deployed using Allaire's ColdFusion Web application server and Compaq's Proliant server. Ableauctions will broadcast live auctions via the Web as well as host silent and charity auctions.
www.ableauctions.com.

Synergy 1.51 Provides Multilingual Interface

(Vancouver, BC) – Catouzer Inc., developer of Synergy and a leading ColdFusion Web application development firm, has released the latest version of its flagship product, Synergy, the ColdFusion-based Web application framework for corporate intranets.

The new version adds Spanish to its multilingual interface of five languages – the others are English, Japanese, French and German – "allowing people to work together, talk together, share information and manage their office environment," according to Wayne Browne, director of product marketing.

Synergy offers security and administrative services and eight fully integrated applications for deploying enterprise intranet solutions.
www.catouzer.com

ALLAIRE ALLIANCE PARTNER

www.cf-training.com

GALILEO

www.coldfusionjobs.net

Employment

Employment

Employment

JDJStore.com

GUARANTEED BEST PRICES FOR ALL YOUR JAVA RELATED SOFTWARE NEEDS

Take a look at our specials this month!

EASTLAND DATA SYTEMS

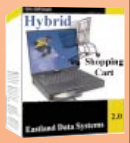
Internet Shopping with Java Shopping Cart

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

. . . . . $294.99

Hybrid Shopping Cart

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



. . . . . $294.99

CGI Shopping Cart

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

. . . . . $294.99

Guaranteed Best Prices

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

Terms of offer:

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

Prices subject to change.
Not responsible for typographical errors.

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

ALLAIRE

ColdFusion 4.0

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

ColdFusion Studio 4.0 . . . . . $354.99
SkillBuilding with ColdFusion Interactive Training CD $284.99

ALLAIRE

HomeSite 4.0

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

HomeSite 4.0 . . . . . $87.99

ALLAIRE

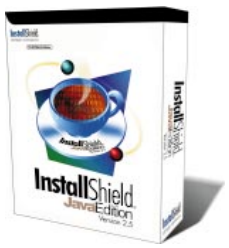
JRun - Live Software

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

JRun $558.99

INSTALLSHIELD

InstallShield Java Edition 2.5



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

InstallShield Java Edition $474.99

PROTOVIEW

Java Enterprise Editions

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

JSuite Enterprise Edition $818.99
JFC Enterprise Edition $818.95
JSuite. $328.99
JFCSuite. $408.99

KL GROUP

JProbe Suite

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

JProbe Profiler w/ Standard Support (inc. JProbe Memory Debugger) . . . $464.99
JProbe Coverage w/ Standard Support $464.99
JProbe Threadalyzer w/ Standard Support $464.99
JProbe Suite w/ Standard Support $934.99

GALILEO DEVELOPMENT SYSTEMS

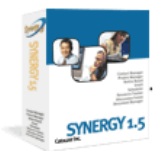
Intr@Vision Foundation

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

Intr@Vision Foundation $3499.99

CATOUZER

Synergy SOHO



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

Synergy SOHO $499.99

CF USER GROUP LISTING

User Group	Contact	e-mail	Phone
NORTH AMERICA			
Canada-Toronto	Kevin Towes	ktowes@PangaeaNewMedia.ca	416-922-1600
Canada- Vancouver B.C.	Cameron Siguenza	cameron@catouzer.com	604-662-7551
Alabama- Birmingham	Jeff Bryant	jeff.bryant@southtrust.com	205-667-5204
Alabama- Huntsville	Tasia Malakasis	tmalakasis@creativeis.com	256.971.9104
Arizona - Scottsdale	Christopher Kracht	cfug@aznet.com	480-839-8239
California- Bay Area	Nathan Dintenfass	ndintenfass@creativeis.com	415-552-9400
	Kelly Ross	kross@creativeis.com	415-793-9301
California- Los Angeles	Bruce Braunstein	bruceb@yahoo.com	818-789-6932
California- Sacramento	Brent Yager	byager@solvix.com	916.456.1972
California- San Diego	John Morgan	johnmorgan@bluestarcorp.com	619-622-1201
California- Southern	Leon Chalnick	lchalnick@prprpr.com	562-491-0212
Colorado - Northern	Robi Sen	robi@granularity.com	970-266-9125
Florida- Tampa	Dale Mulert	dale@eng-tech.com	888-382-1900
Florida- Orlando	April Fleming	afleming@ist.ucf.edu	407-658-5060
Georgia- Atlanta	Cameron Childress	cameronc@mcrae.com	770-460-7277
Illinois- Chicago	Bob Burns	bob-burns@mediaone.net	630-961-6261
Maryland- Baltimore	Steve Drucker	sdrucker@figleaf.com	202-797-5477
Maryland- Bethesda	Michael Smith	michael@teratech.com	301-424-3903
Massachusetts- Boston	Steve Casco	steve@ateaze.com	781-641-1661
Minnesota- Minneapolis	Dan Chick	chickd@fastcart.com	612-561-1500
Missouri- Kansas City	David Mertz	dbmertz@idiservices.com	913-248-8814
New Jersey- Murray Hill	Cindy Pomarlen	cindy_pomarlen@hotmail.com	973-540-9672
	Charlotte Wilbush	wilbush@skytechnologies.com	908-582-3000
New York- NY City	Michael Dinowitz	mdinowit@houseoffusion.com	212-647-8901
New York- NY City	Bob Siegel	chairman@cfsig.org	718-768-3245
New York- Syracuse	Dave Santeramo	dsantera@zdnetmail.com	607-756-0393
North Carolina- Charlotte	Glynn Wallace	gwallace@webserve.net	704-556-7482
Ohio- Columbus	Angelo McComis	angelo@compuserve.com	614-538-4539
Ohio- Cleveland	Jim Hoch	jhoch@newchanneltech.com	330-220-1558
Oregon- Portland	William Cupps	billc@dev-tech.com	503-294-4488
	Paul Williams	paulw@isitedesign.com	503-605-9100
South Carolina- Greenville	Patrick Little	patricklittle@home.com	864-244-8965
Texas- Austin	Dave Williams	david.williams@ci.austin.tx.us	512-499-2835
	Miki Hardisty	miki.hardisty@the401k.com	512-310-9650
Tennessee- Nashville	James Pecora	jpecora@radsoltn.com	615-269-4493
Texas- Dallas (Fort Worth)	Patrick Steil	pmsteil@imailbox.com	940-321-6162
Texas- Houston	Andy Dubinsky	andy@netfast.net	713-623-4366
Texas- San Antonio	Chris Montgomery	monty@astutia.com	210-490-3249
Utah- Salt Lake	Scott Hefron	sheffron@pcs-link.com	801-558-2844
Virginia- Charlottesville	Steve Nelson	m@secretagents.com	804-293-2060
Virginia- Hamden Roads	Raymond Camden	jedimaster@creativeis.com	757-318-7042
Virginia- Northern	Steve Drucker	sdrucker@figleaf.com	202-797-5477
Washington D.C.	Steve Drucker	sdrucker@figleaf.com	202-797-5477
Washington- Seattle	Bob Schneider	bobs@qinteractive.com	425-974-1000
INTERNATIONAL			
New Zealand, Auckland	Amanda Irvine	amanda.irvine@delta.co.nz	09-309 9400
Australia- Melbourne	Marc Dimmick	marc@tved.net.au	613-9670-2055
Australia- Sydney	Brent Pearson	b.pearson@morganbanks.com.au	
Belgium- Brussels	Francois Lamotte	f.lamotte@switchon.be	
	Denis Wauthy	d.wauthy@switchon.be	
Europe (Central)	Sven Slazenger	slazenger@interlake.net	49-7542-1204
The Netherlands	Simon Slooten	slooten@prisma-it.nl	+31 (0)10 458 7148
Singapore	Chou Hao Chan	chiouhao@intracomm.com	
South East Asia	Roydean Osman	roydean@gtemail.net	972-523-0888
Taiwan- Taipei	Jack Jair	cfmaster@leetide.net	+886-2-2346-7525
United Kingdom	Paul Underwood	paul@cfug.co.uk	+44 (0) 171 589 4500

Spectra *continued from page 66*

While the PC era is extremely mature, the Web era is just beginning. In the early phases technology decisions were driven principally by early adopter developers. The focus was on tools, and most things were built completely from scratch. As the market matured, the tools turned into core platforms. For example, while early-on servers like ColdFusion were treated like tools for one-off projects, this server technology increasingly added more and more core capabilities, making it a true foundation or platform. What is now the "Web application server" marketplace resembles core platforms – such as databases, operating systems and networks – more than tools. The price points and additional capabilities reflect this.

Indeed, the "winners" in this emerging Web-platform landscape will be the companies who, like their counterparts in earlier computing eras, are able to supply the tools, platforms and solutions for making companies successful with the Web. As in the past, they'll be those who've built incredible and loyal developer communities capable of working with and benefiting from their platform.

A more direct issue with Allaire Spectra is understanding its relationship to ColdFusion. The first thing to remember is that Allaire Spectra is a solution, not a tool; the second, that ColdFusion is the platform on which Allaire Spectra is built and the tool used to implement and extend the system. Indeed, Allaire Spectra empowers ColdFusion developers with an incredible foundation of services and tag libraries for building large-scale Web sites while simultaneously empowering the rest of your organization to use the Web productively on a daily basis.

Allaire Spectra includes over 250 new tags! All of them are exposed to developers for building their own custom applications and solutions using ColdFusion. These tags include a workflow system, a browser detection engine, an XML content management database, a user profiling and personalization engine, and a publishing and caching engine. In addition, Allaire Spectra ships as source code under a community source license. We believe that to support our massive developer base we need to bring them directly into our product and code for peer review and mutual evolution. From this foundation there is an incredible amount to learn for every developer seeking to take their advanced ColdFusion development skills to a new level – and the rest of their organization with them.

Introducing **Allaire Spectra**

You say you want an evolution?

BY
JEREMY ALLAIRE

Introduced in late July 1999, Allaire Spectra is a packaged system for content management, e-commerce and personalization. As a new product line, Allaire Spectra represents a significant shift in the evolution of Allaire as a leading supplier of software for the Web, creating enormous opportunities for companies using Allaire software and developers building on our platform. What follows is less of a product overview and more of a background on Allaire Spectra and what role it will play in companies' adoption of the Web.

Initially, many people wonder what the foregoing description of Allaire Spectra means. What is a packaged system? Why content management, e-commerce and personalization? Is this a tool or an application? How would I use it as a developer?

Packaged system means that we've built the solutions to common problems our developers have encountered over and over again in dynamic publishing systems, rights management and user security systems, and analysis and reporting tools, and packaged them into frameworks that can be quickly and easily applied in your own business. It's a system rather than an application. An application is monolithic, prebuilt and out-of-the-box. A system is something you build on and customize for your own purposes.

Allaire focuses on three core solutions that represent what we found to be the most common things people want to make the Web more strategic to their businesses. First, we saw that companies who wanted to use the Web more effectively to communicate, conduct commerce, and deliver sales and service offerings were hampered by inadequate infrastructure for managing their Web content. While moving to dynamic sites using databases and templates (such as plain old ColdFusion) was a good start, true enterprise-wide adoption or the creation and management of large-scale sites required much more infrastructure and higher-level solutions.

Some of these things included workflow, managing content in XML and having rich, roles-based security systems across their Web site. Others included adding richer capabilities like personalization, and rich analysis and reporting tools for understanding what was happening in the Web business.

E-commerce was one of the most common things our customers were doing, and the major challenges they faced weren't building shopping carts or order-processing tools (these are actually quite easy to do with ColdFusion on its own), but effectively melding their existing business processes (such as creating a product promotion or handling a customer service request) into the Web, and extending their business out to Internet business partners in an open and secure fashion. For this we created rich tools for modeling and managing common business processes, and for empowering business users and managers – who are truly the key drivers in an e-commerce site – to get their work done without ongoing IT intervention.

We also created a business intelligence framework that allows developers to easily track user interactions within your Web site, generating custom reports and personalization rules. Finally, we created a model that makes it easy to conduct secure, business-to-business commerce using XML and Web syndication.

Allaire Spectra is a solution, not a tool. This shift toward supplying more packaged solutions is part of a broader shift occurring with Allaire and the market in general. As more and more companies move their businesses to the Web (internally and externally), we're starting to see a new class of decision makers involved in driving Web investments, and we're also seeing that many companies don't want to start at ground zero. The work we've done as a developer community in the past four years has created an enormous amount of knowledge about what it takes to create and manage a successful Internet business. Allaire Spectra attempts to encapsulate and package that knowledge into a solution that can easily be implemented using ColdFusion.

The other shift is for Allaire. As the market matures, Allaire is shifting from being a point-provider of Web development tools to becoming a major player in the Web software industry, or what some people call a platform company. As part of this shift, Allaire is building a full suite of products that spans tools, application servers, management products and packaged solutions. Combined, this platform gives companies a strong foundation on which to build their Internet business.

Microsoft, for example, supplies core platforms – such as their operating systems – and key back-office components – such as SQL Server. To drive the success of their core platforms, they supply great tools that make it easy to take advantage of the underlying services their platform provides. Furthermore, they supply common solutions built on, and extensible with, this foundation. For example, in the PC era the killer "solutions" products were the Office productivity applications – solutions for office workers to effectively work with office information. Other examples include enterprise server solutions; solutions for enabling collaboration, such as Microsoft Exchange and Outlook; and solutions for content management and e-commerce, such as Microsoft SiteServer.

What made this so attractive to developers was that there's so much to leverage in the Microsoft platform. The operating system provides a rich foundation and programming model for Windows applications. Tools are available for custom solutions, and they're leveraged to extend and deepen the packaged solutions.

continued on page 65

ABOUT THE AUTHOR

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