<BF on CF>: Expect the Unexpected
Bugfree code – a lofty goal, but attainable

Ben Forta

CFDJ Feature: Spectra: The New Application Standard
Its abstraction of work flows and functionality allows developers to rapidly rework their applications

Anthony Krinsky

Server to Client WDDX: Send ColdFusion queries to JavaScript for use in <SELECT> tags

Matthew Boles

CFDJ Feature: Spectra and the Metadata Service
A way to index your content that's powerful and easy

Raymond Camden

CFDJ Feature: A Help Engine for Web Users
How to win friends and delight users

Hal Helms

Advanced CF: Architecting Enterprise Level CF Apps
One step closer to emulating the world

Ed Donohue & Benjamin Elmore

CF Automation: We're Off to See the Wizard
Developing code wizards in CFML

Steven D. Drucker & Robin Dilley
Creating Subforms

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

By

Mark
Czyk

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.

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 83-98. 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 onClickQ 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 addltQ 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 addltQ 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 buildFinalDisplayQ 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.
Allaire, AT YOUR Service.

YOU’RE BUILDING POWERFUL APPLICATIONS WITH COLDFUSION.

Meet the global organization that stands behind you.

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

Instructor-led training programs
to get you up to speed fast

www.allaire.com/training
Fast Track to HTML (2-day course)
Fast Track to ColdFusion (3-day course)
Advanced ColdFusion Development (2-day course)

Allaire training has helped thousands of developers jump-start their projects — and their careers. You’ll learn by doing with a dynamic combination of lectures, demonstrations, hands-on labs, and take-away materials. Access classroom training in your area, or at your site, throughout the year. There’s no better foundation for ColdFusion developers.

DevCenter gives you the resources to ignite your productivity

www.allaire.com/devcenter

Allaire has created a rich, web-based resource center for developers using ColdFusion — available 24-hours a day and free of charge. Here you’ll find custom tags, sample applications, technical tips, documentation and third-party tools you can use to accelerate your own development process. You’ll also find links to the ColdFusion Support Forum where you can take part in the lively exchange among customers, Allaire employees, and advanced Team Allaire developers.

Technical support and consulting services

www.allaire.com/support
(or call 888-939-2545 or 617-761-2000 outside of the US)

Allaire offers a wide range of fee-based telephone support programs tailored to the needs of professional web developers and enterprise development organizations. We also offer on-site consultants who have the expertise in ColdFusion application architecture, database design, performance testing, and environmental/configuration considerations. Our consultants will help you discover and eliminate factors that could impact the success of your site.

At Allaire, we’re at your service.
Helping you increase your productivity and accelerate your time to market.

After all, it’s a net-paced world.

ColdFusion is a US registered trademark, and Allaire, Allaire logo, and the ColdFusion logo are trademarks of Allaire Corporation. All other brands and products are trademarks of their respective holders. © 1998 Allaire Corp. All rights reserved.
Event Title:
| Second Event |

Event Sponsor(s):
- Still Another Second Event Sponsor
- Yet Another Second Event Sponsor
- Another Second Event Sponsor

Submit Event!

FIGURE 2: The insert form

<table>
<thead>
<tr>
<th>eventID</th>
<th>eventtitle</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>First Event</td>
</tr>
<tr>
<td>8</td>
<td>Second Event</td>
</tr>
<tr>
<td>9</td>
<td>Third Event</td>
</tr>
<tr>
<td>(AutoNumber)</td>
<td></td>
</tr>
</tbody>
</table>

FIGURE 3: The Event table

Line 33 simply returns focus to the now empty "eventsponsor$textbox" 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 addLTQ function; Line 41 of the addLTQ 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 deleteFromArrayQ 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 "eventsponsor$textbox" 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 by creating a content creator array.

You need a web host who understands Cold Fusion the way you do

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

One of the most advanced monitoring systems to insure Cold Fusion availability
Technicians who actually program in Cold Fusion
Powerful Dual Pentium 350 servers with 256MB of RAM
DS3 and T1 connectivity with multiple backbone providers
24x7 system monitoring
30 day money-back guarantee
Telephone support M-F 9am - 6pm and Sat 10am - 6pm

100 MB disk space and 3GB traffic
Multiple CFX and CF tags available
2 free datasources
SQL Server available
10 POP 3 mailboxes
Free Cybercash and SSL
Daily graphical stats

http://www.virtualscape.com - 212.460.8406
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 addlt() 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.
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 it 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.
Building E-business systems isn't like it used to be.

Applications are complex and distributed involving many components and connecting interfaces. They are composed of "emerging" technologies and are developed in Internet time.

They are expected to be business critical in reliability.

WebLoad is the only Web application testing solution that unifies load, performance and functional testing into a single process for shortened development cycles and unmatched verification of Web application scalability and integrity prior to deployment. For more information, visit www.radview.com, or call 1-888-RADVIEW (723-8439).

©1999 RadView Software, Incorporated. All rights reserved. The RadView logo, WebLoad, and the WebLoad logo are either registered trademarks or trademarks of RadView Software, Inc.