

Domino and JavaScript: Dynamic Partners (Part III)

by David MacPhee, Karen Hobert and Russell Lipton

[Editor's note: This article resides in "Iris Today", the technical Webzine located on the <http://www.notes.net> Web site produced by Iris Associates, the developers of Domino/Notes. This is the final article in a series of three on how to use Domino and JavaScript to design dynamic Web applications. This third article explains how to use JavaScript to provide address book functionality for Web users, and how to easily incorporate context-sensitive help documents in Web-based applications.

The [first article](#) introduced you to Java and JavaScript, and discussed some guidelines for when and where you might want to use JavaScript in your applications. The [second article](#) explained how to do field validation with JavaScript by walking through the Domino.Applications SiteCreator application.]

Implementing Address Book Lookups and Context-Sensitive Help

While we cannot predict JavaScript's long-term direction, we can safely say that its usefulness is growing, as we continue to author this series of articles. For the moment, JavaScript is the language of choice for controlling the behavior of Web clients.

In this third article, we'll build on the simple browser detection and field validation routines described in [Part Two](#), and show you how to create interactive dialog boxes for your Web applications by using Notes forms, pass-thru HTML and generation of JavaScript at the browser. We'll specifically look at dialog boxes that add most of the functionality and look of the native Domino Public Address Book for the Web, as well as context-sensitive help.

To do this, we'll walk through a sample database that contains two Domino/JavaScript goodies: the Address Box dialog and the Context-Sensitive dialog. First, we'll introduce you to our design strategy for these dialogs, and then discuss in detail how each one looks and works.

The best part is that you can customize the sample database, and use the same techniques discussed in this article, to cover almost any situation where you want to implement interactive dialog boxes and their associated data for the Web.

Our design strategy

When we began designing the Address Box dialog and Context-Sensitive Help dialog, our main goal was to make data entry and display as convenient as possible for users. This meant that we needed to take the different types of browsers into consideration.

Although both Microsoft and Netscape offer powerful, useful features directly within their browsers, we decided to focus only on what is held in common and, more specifically, on the manipulative leverage given by JavaScript. Even so, we came upon important browser distinctions that forced us to design our application along two somewhat divergent codepaths. For example, we rely heavily on HTML "selects" to update field contents on the browser. Netscape Navigator, Netscape Communicator and Internet Explorer 4 (IE4) all support these "selects." However, IE3 does not support this.

Consequently, we came up with the following design strategy:

1. When an application is launched, the "About Document" transmits a small chunk of JavaScript that queries the target browser for its type and version. (Remember that [Part Two](#) of this series discussed the code for detecting types of browsers.)
2. Control is redirected along the appropriate codepath (for instance, an "IE4" path), keeping in mind that much code is shared between the major browser types.
3. LotusScript agents then dynamically build and transmit back to the browser the required JavaScript for a given browser type, consistent with the rest of the application logic coded within Notes forms, fields and views.

Another part of our design strategy was that we wanted the database to work on both Domino 4.5 and 4.6. This meant that we needed to use more manual coding, because the new options available through infoboxes and menus in Domino 4.6 are ignored when the application runs on a Domino 4.5 server. For example, Domino 4.6 now supports multiple buttons on the page -- you just need to enable the "Web access: Use JavaScript when generating pages" on the database properties. However, you need to manually enter the JavaScript code for the multiple buttons to work when served from Domino 4.5.

Now, let's look at how the sample database looks and works.

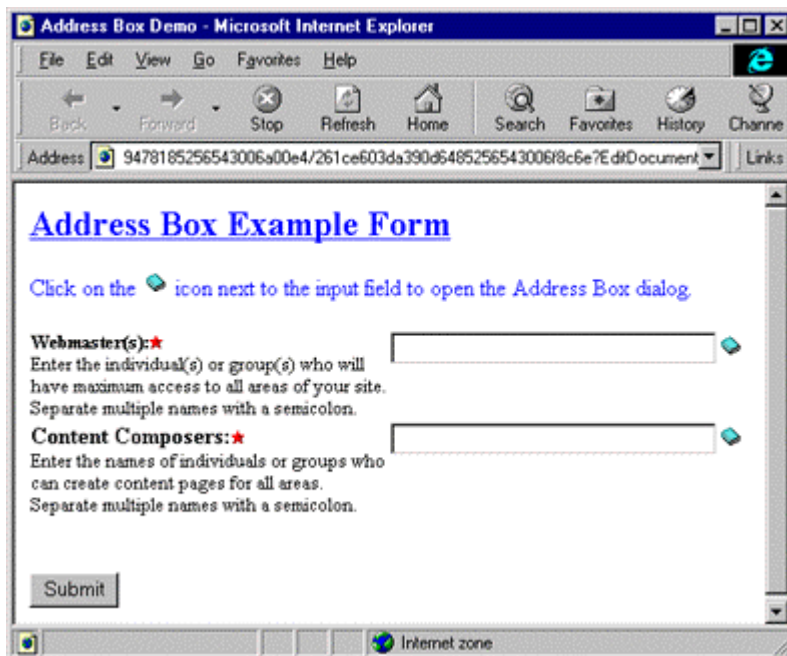
Downloading the sample database


You can try out the techniques described in this article by downloading the self-extracting sample database (available from Iris Today at <http://www.notes.net>).

Note: If you decide to customize the sample database, make sure that the doclink in the "About" document still links to the only document in the "DemoDocument" view. Also, make sure to copy the AddressBox\$nn and AddressBox\$ie3 forms into your database design.

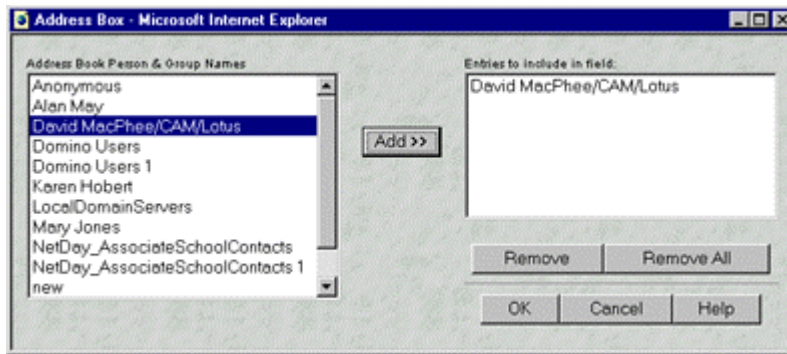
How the Address Box dialog looks

Our Address Box dialog allows Web users to do lookups in the Public Address Book, with most of the same functionality and look that they would have on a Notes client. As stated earlier, we needed to take browser distinctions into consideration when designing the dialog. The initial Address Box form looks about the same on all the browsers, as shown below. It's the functionality behind the form, and the dialog that it brings up that is different, depending on the type of browser.



The screenshot shows a web browser window titled "Address Box Demo - Microsoft Internet Explorer". The address bar displays a long URL. The main content area has the heading "Address Box Example Form" in blue. Below the heading, there is a blue instruction: "Click on the  icon next to the input field to open the Address Box dialog." There are two input fields. The first is labeled "Webmaster(s):★" and has a description: "Enter the individual(s) or group(s) who will have maximum access to all areas of your site. Separate multiple names with a semicolon." The second is labeled "Content Composers:★" and has a description: "Enter the names of individuals or groups who can create content pages for all areas. Separate multiple names with a semicolon." Both input fields have a small book icon to their right. At the bottom left, there is a "Submit" button. The status bar at the bottom shows "Internet zone".

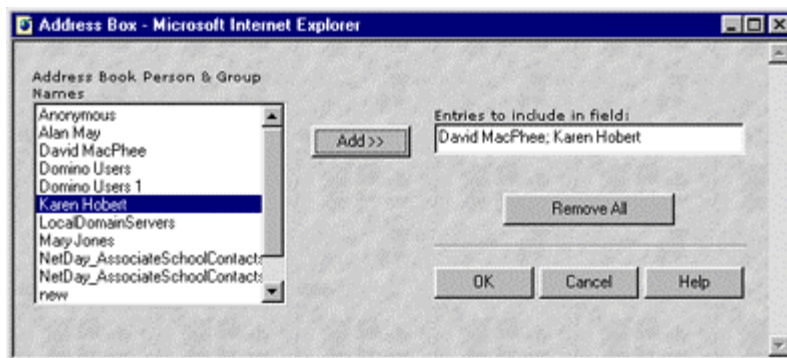
When you click on the book icon next to the input field, Domino displays the appropriate Address Box dialog according to your browser type. The following dialog displays for Netscape Navigator, Netscape Communicator and IE4 users.



The buttons work like this:

- Add>> -- Moves names from the left box to the right. It can move multiples, but checks for uniqueness (sorts them too).
- Remove -- Removes a selected name from the right box.
- Remove All -- Removes all entries from the right box.
- OK -- Copies the contents of the right box back to the Web page field.
- Cancel -- Closes the window without an update.
- Help -- Opens a context-sensitive help window (covered in the next section).

If you're using IE3, you'll see the following dialog instead.



The above Address Box dialog works slightly differently than the first one. Notice that the right box is now a text field, and the Remove button is gone.

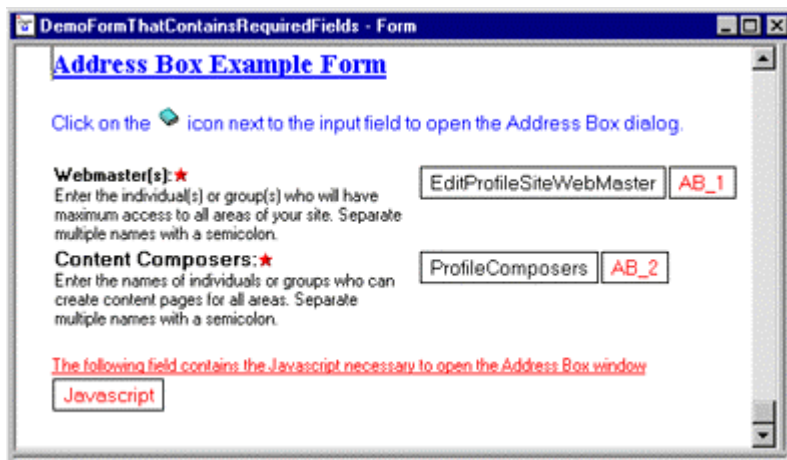
The main difference between the dialogs is that the first one uses a <SELECT> box for the entries. The SELECT is like an array from which we can remove individual, selected elements. However, Microsoft's Jscript implementation of IE3 doesn't allow us to update <SELECT> boxes after they are initially populated. So, we needed to create a separate dialog for IE3 in which the right box is <TEXT>. We are then forced to update an <input type=text> object, which is a single entity. We can only "Remove All" of the object contents, not the individual names. Therefore, "Remove" has no context here.

How the Address Box dialog works

Now, let's take a closer look at how we created the Address Box dialog. Instead of using subforms or hide-whens to combine the browser differences into one form, we decided to use two separate forms:

- AddressBox\$nn -- The form for Netscape Navigator (NN3), Netscape Communicator (NN4) and IE4 browsers.
- AddressBox\$ie3 -- The form for IE3 browsers. This separate form is a bummer, but necessary.

As shown below, these forms contain fields that are populated by the address book (EditProfileSiteWebMaster and ProfileComposers), fields that display a book icon for bringing up the address book (AB_1 and AB_2), and a hidden JavaScript field that contains the actual code for opening the address book.



The screenshot shows a web browser window titled "DemoFormThatContainsRequiredFields - Form". Inside, the form is titled "Address Box Example Form". It contains a blue instruction: "Click on the icon next to the input field to open the Address Box dialog." Below this are two sections. The first is "Webmaster(s):" with a red star icon, a text input field, and a button labeled "AB_1". The second is "Content Composers:" with a red star icon, a text input field, and a button labeled "AB_2". At the bottom, there is a red text label: "The following field contains the Javascript necessary to open the Address Box window" and a button labeled "Javascript".

The three pillars of the lookup process

Our JavaScript techniques are held together by three cooperating pieces of code. While conceptualizing an elegant solution for dialog displays and address book lookups was challenging, the code itself is remarkably simple:

1. A common Notes field that can be placed on any form containing a reference to a second field. The AB_1 and AB_2 fields fulfill this function for us.
2. A JavaScript function that displays a small icon which, when activated, opens a new browser window and displays the address box with the contents of the address book. The code for opening the address book is tied to the hidden JavaScript field, while the icon code is attached to the AB_1 and AB_2 fields.

Here's the code within the "Javascript" field:

```
dbPath:=@ReplaceSubstring(@ReplaceSubstring(@Subset(@DbName;-1); " "; "+" ); "\\"; "/" );
@NewLine +
"<script language=javascript>" + @NewLine +
"browser = (navigator.appName == 'Microsoft Internet Explorer' &&
navigator.appVersion.indexOf('3') > -1) ? '$ie3' : '$nn';" + @NewLine +
"function addressBox( fieldName ) {" + @NewLine +
"  abURL = \" + dbPath + \"/AddressBox\" + browser + \"?OpenForm\";" + @NewLine +
"  addressField = fieldName;" + @NewLine +
"  wheight = ( browser == \"$nn\" ) ? \"250\" : \"200\";" + @NewLine +
"  wwwidth = ( browser == \"$nn\" ) ? \"620\" : \"550\";" + @NewLine +
"  opts = \"width=\" + wwwidth + \",height=\" + wheight;" + @NewLine +
"  popupWin=window.open( abURL,\"abWindow\", opts);" + @NewLine +
```

```
"}" + @NewLine +
"</script>"
```

The code below, attached to the AB_1 field, displays the "abook.gif" icon" and executes an HTML link, invoking an addressBox function with the addressField's value as a parameter.

```
REM "The addressField variable should contain the name of the field that will receive the output of the
Address Box dialog";
addressField := "EditProfileSiteWebMaster";

REM "Don't change the following string";
"<a href='\"javascript:addressBox(\" + addressField + \"')\"><img src=\\icons\\abook.gif' border=0>"
```

3. A Notes form that performs the needed lookups and generates JavaScript code for return to the browser. This is where the bulk of the work is done. The AddressBox\$nn and AddressBox\$ie forms play this role.

Here, we display the main JavaScript code that makes this application work for the Netscape browser. You can also find the code entered directly onto the AddressBox\$nn form, where the scripts are passed through the Domino server for execution by the browser:

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

```
/* This Domino.Applications JavaScript code is Copyright (c) 1997 Lotus
 * Development Corporation, a subsidiary of IBM Corporation, all
 * rights reserved. None of this code may be reproduced in any form
 * without permission. Code written by David MacPhee of
 * www.baddogbad.com Permission: You may use this code as long as
 * this comment section is included intact.
 */
```

```
//...get a pointer to the field we will update...
addressFieldName = window.opener.addressField;
addressField = window.opener.document.forms[0][addressFieldName]
```

```
//...and the two SELECTs on this form...
theList = document.forms[0].theList;
tmpList = document.forms[0].tmpList;
```

```
//...initialize the temporary SELECT on this form (the object of the Add>> button's desire)...
```

```
function init ( list ) {
    theList.options[0] = null;
    list = addressField.value;
    beg = 0;
    del = ",";
    end = list.indexOf( del )
    if ( end == -1 ) end = list.length;
    i = 0; j=1;
    while ( true ) {
        theName = list.substring( beg, end );
        if ( theName != "" ) {
            tmpList.options[i].text = trim(theName);
            i++; j++;
            tmpList.options.length = j;
        }
    }
}
```

```

    }
    beg = end + 1
    if ( end == list.length ) break;
    end = list.indexOf( del, beg)
    if ( end == -1 ) {
        end = list.length;
    } else {
        if ( del != ';' ) end--;
    }
}
tmpList.options.length = i;
}

//...BUTTON handler for the OK click...
function okClick() {
    if ( tmpList.options.length > 0 ) {
        addressField.value = tmpList.options[0].text;
        for ( i = 1; i < tmpList.options.length; i++ ) {
            addressField.value += "; " + tmpList.options[i].text;
        }
    } else addressField.value = "";
    self.close();
}

//...BUTTON handler for the REMOVE click...
function removeClick() {
    tmpLength = tmpList.options.length;
    for ( i = 0; i < tmpList.options.length; i++ ) {
        if ( tmpList.options[i].selected ) {
            tmpList.options[i] = null;
        }
    }
}

//...BUTTON handler for the REMOVE ALL click...
function removeAllClick() {
    tmpList.options.length = 0;
}

//...BUTTON handler for the ADD click...
function addClick() {
    tmpLength = tmpList.options.length;
    for ( i = 0; i < theList.options.length; i++ ) {
        if ( theList.options[i].selected ) {
            newName = theList.options[i].text;
            addIt = true;
            for ( j = 0; j < tmpLength; j++ ) {
                if ( tmpList.options[j].text == newName ) {
                    alert ( newName + " is already in the list." );
                    addIt = false;
                    break;
                }
            }
        }
    }
    if ( addIt ) {

```

```

        tmpList.options.length = ++tmpLength;
        tmpList.options[tmpLength-1].text =
            newName;
    }

    }

}

sortList (tmpList)
}

//...a function to remove leading blanks, which could cause problems...
function trim(strIn) {
    strOut = strIn;
    for (var t = 0; t < strIn.length; t++) {
        if ( strIn.substring( t, t+1 ) != " " ) {
            return (strOut);
        } else {
            strOut = strIn.substring( t+1, strIn.length);
        }
    }
    return (strOut)
}

//...a sort to keep the tmpList in order...
function sortList (arrayIn) {
    for (i = 0; i <= arrayIn.length; i++) {
        j = i;
        for (k = i; k < arrayIn.length; k++) {
            if (arrayIn[k].text < arrayIn[j].text) {
                j = k;
            }
        }
        if (j > i) {
            l = arrayIn[j].text;
            arrayIn[j].text = arrayIn[i].text;
            arrayIn[i].text = l;
        }
    }
}

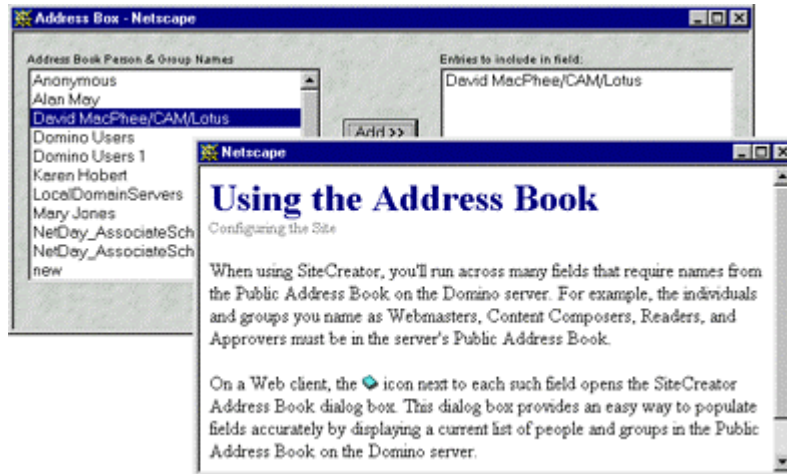
//...call the init() function upon load...
init ();
// end -->
</SCRIPT>
<!-- hide the Domino generated Submit button><---->

```

That's it! You can customize this design by using the DemoFormThatContainsRequiredFields form as an example. Simply copy the AB_1, AB_2 and Javascript fields. You can change the name of the field that's populated from the address book. Just make sure to change the addressField variable in the AB_1 function to that new field name (that is, replace EditProfileSiteWebMaster with the new field name).

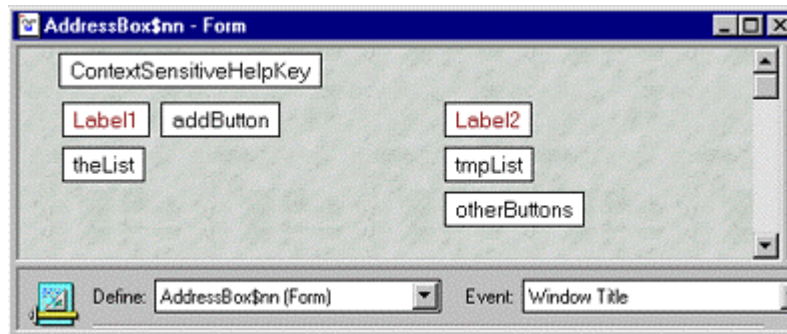
How the Context-Sensitive Help dialog looks

We used a related, though simpler approach to retrieve and display context-sensitive Help documents. When you click the Help button in the Address Box dialog, its help document opens in its own window.



How the Context-Sensitive Help dialog works

You can take a more freeform approach to designing this Context-Sensitive Help dialog than with the Address Box. Use our sample as just a guide for achieving your own goals. Here's the help portion of the "AddressBox\$nn" form:

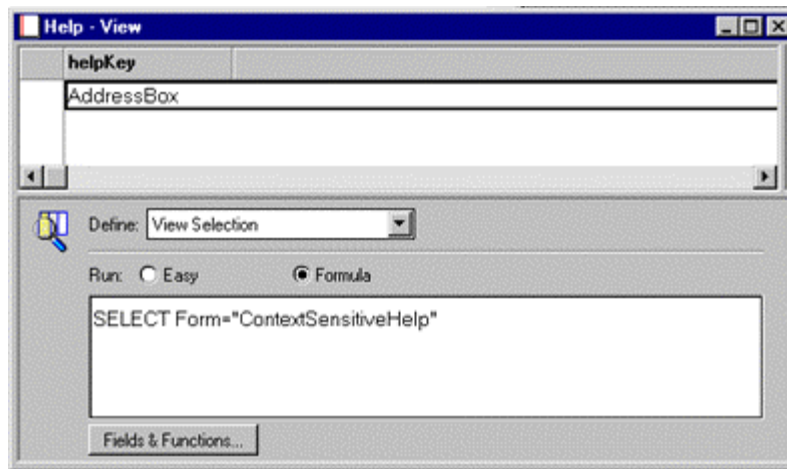


The ContextSensitiveHelpKey field contains the key of a document in the Help view related to the AddressBox dialog. In this particular case, the key value is AddressBox.

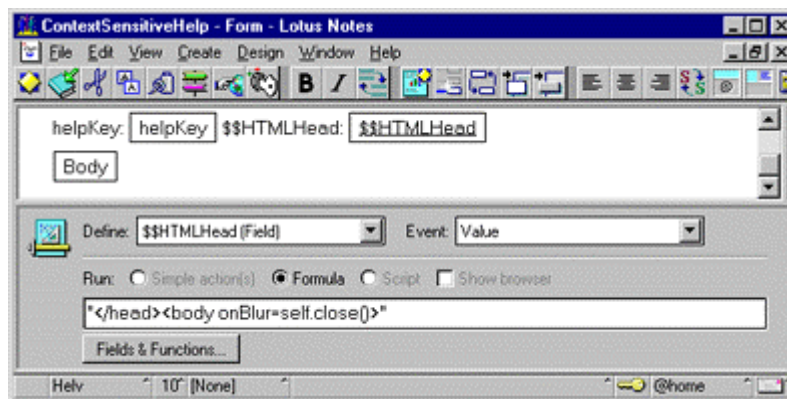
The otherButtons field contains the JavaScript code needed to open a window containing the help document. The URL for locating the help document is constructed within the form by this code snippet:

helpURL := "/" + dbPath + "/Help/" + ContextSensitiveHelpKey;

The Help view shown here contains documents sorted by the ContextSensitiveHelpKey:



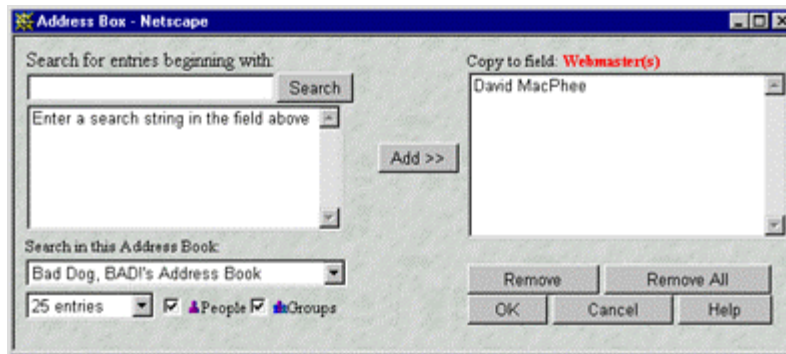
Our Help form looks like the one shown below. It is simple, but effective. Make sure that you include the \$\$HTMLHead field (or code this in the HTML Attributes of the form if you are using 4.6). This way, the help window will close automatically whenever the user clicks away from it in the browser.



A peek into the future

The code included with this article is also incorporated in [Domino.Action](#) and the [Domino.Merchant 2.0 Server Pack](#) (Club Med) applications. Because it uses the @DBLookup function, the Address Box dialog is limited to 15k of names and groups. This dialog is not designed to accommodate cascading address books (although it isn't too hard to convert for this purpose). All but the very large address books can be handled.

For the Domino.Merchant Server Pack, we included a more robust implementation that can accommodate any size address book (as well as cascading address books). Describing this implementation would be needlessly complicated for the purpose of this article. You can visualize our fundamental approach by looking at its new display:



A fresh look at using Notes as a development tool

One thing we've tried to show you in this article is that Notes can be a powerful tool for developing interactive Web applications. In fact, you can use Notes forms, fields and views to integrate the browser directly into the application. This is by direct contrast with the way that browsers are treated by other Web applications, that is, as simple presentation targets. Even though that model is extended somewhat by the ability to compute JavaScript within the browser, the browser is still "walled off" from other application components. By using native Notes to compute even values that could be computed on the browser (for instance, variables), you gain fine-grained control over application design, logic, management and performance.

In the case of forms, you can intersperse pass-thru HTML and JavaScript code with Notes fields, which allows you to take advantage of Domino's execution order. You can treat fields logically as though they were variables (that is, as containers that can return an arbitrary value based on a computation), knowing that Domino will compute fields before processing other text on the form and rendering it to the browser. Once the fields have been processed, the HTML, JavaScript and computed field values can be passed through the Domino server for further execution on the browser.

Notes views enable you to build JavaScript arrays (where the contents of the array are the values within a row of the view) or hold parameters that will later be used as Java applet parameters by the browser (allowing you to manage parameters within Notes rather than at the browser). That is, views can serve as the repository for generating code dynamically on the browser.

So, we can retrieve into Notes any data that the user enters at the browser. Then, we can dynamically massage, manipulate, store and redirect that data back to the browser with superb interactivity. This includes generation of JavaScript itself on the browser. The browser (assuming, of course, it is JavaScript-enabled) still executes JavaScript code as well as HTML tags, but under the control and management of Notes. We gain this power without having to concern ourselves with anything other than common browser features supported by the two main players (Microsoft and Netscape).

These techniques will enable you to create a more refined, flexible and user-centered interface for your Web-centric Notes applications.

ABOUT DAVID AND KAREN

David MacPhee and Karen Hobert of BadDogBad are long-time Notes developers who have worked closely with Lotus to design and build Web interfaces for Domino.Applications. Bad Dog, BAD! is a Lotus Business Partner that has developed the SiteCreator Web interface for every release of Domino.Action, Domino.Merchant Domino.Broadcast and Domino Intranet Starter Pack. They were a major contributor on the Domino 4.6 WebAdmin Tool and every release of the RealTime Notes product. You can reach the dogs at webdog@baddogbad.com or visit their site at <http://www.baddogbad.com>.

Copyright 1998 Iris Associates, Inc. All rights reserved.