

Creating field help for your Domino applications (part 1)

by Mark Gordon and Micah Blalock

[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 and Notes. This is the first article in a two-part series on how to create context-sensitive field help for your Domino applications. You'll learn how you can provide field help in a separate help frame, basic field help that appears when users tab into a field, and pop-up help. The next article will focus on how to create reusable help, and look at an alternative field help solution for Netscape 4.x browsers.]

By now, forms are a pretty familiar sight on the Web. Seeing field help for those forms, on the other hand, is not so familiar. Some Web sites may show the Web equivalent of Notes field help: a single line of help text that displays in the status bar when you enter a field. But chances are good that you haven't noticed this help, because the status line in your browser is so small! In this article, we will show you a technique for making field help show up in any font and size you like so that your users can't miss it. Like the status bar, your field help won't scroll off the screen. But unlike the status bar, *all* your users will see your help text! And, they'll also be able to click on a "More Info" link to see a window pop up instantly (no server hit) with a more in-depth explanation.

This is the first article in a two-part series that will describe different techniques for adding field help to your applications. In this series, you'll learn techniques for creating:

- No-hassle help frames -- help text that shows up in a small help frame at the bottom of your form. The technique allows you to easily incorporate this help frame without redesigning your entire application to account for frames. And, just as importantly, the help frame doesn't hassle your users because it eliminates the usual URL and bookmarking problems that most framed pages cause.
- Basic field help -- field help that displays when users tab into a field, or move the mouse over a field label
- Pop-up help -- a separate help window that pops up when longer help descriptions are necessary
- Reusable help -- help text that you can reuse for multiple fields on your application. The technique works by storing the help text separately. You'll also learn how to use Domino to generate portions of the JavaScript for you.
- Alternative help for Netscape 4.x browsers -- a work-around for a Netscape 4.x tabbing problem, which causes the tab order to stop working properly when JavaScript is used in field events. This technique generates different JavaScript calls for different browsers.

This time, we'll focus on the first three techniques for creating no-hassle help frames, basic field help, and pop-up help. In the next article, you'll learn about the last two techniques. All the examples we'll discuss are working examples you can try by downloading our sample database.

The article is written with the experienced Domino Web developer in mind. We will do our best to explain the JavaScript constructs we use along the way, so that Domino developers with little or no JavaScript experience can learn some JavaScript and some of the ways to leverage the power of Domino with JavaScript.

Note: If you want to implement field help in your application before reading about it, see the sidebar "[Quick steps to adding field help to your applications.](#)"

Downloading the sample database

You can try out the techniques described in this article by downloading the following self-extracting database (211Kb):

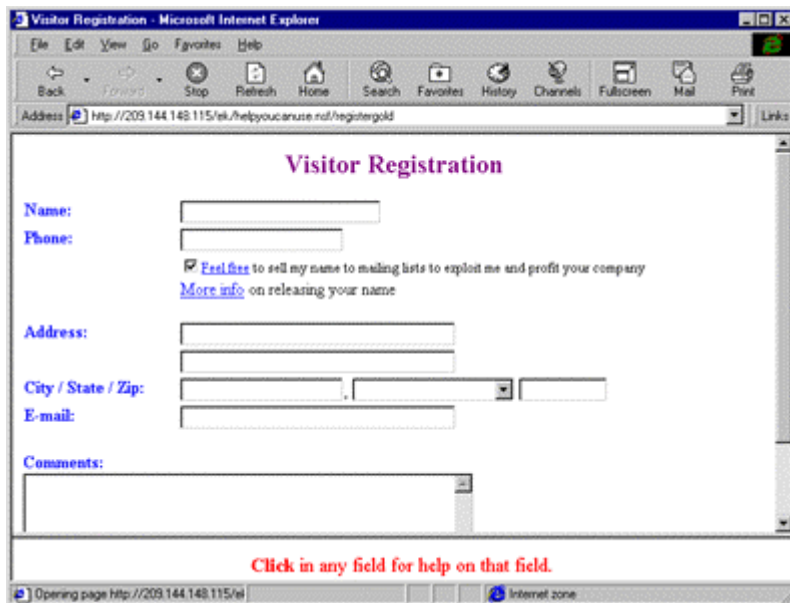


fieldhp2.exe

You can refer to this database as you read through the rest of this article, and then use it to create field help in your own applications. For information on how to use this database to create your own field help, see the sidebar "[Quick steps to adding field help to your applications.](#)"

Displaying field help on a Web form

Let's start by walking through a Web form that uses JavaScript to create context-sensitive field help. This registration form should look much like many others you've seen, with the exception of the frame at the bottom of the form. Notice that the frame contains the red text "Click in any field for help on that field."



Visitor Registration - Microsoft Internet Explorer

File Edit View Go Favorites Help

Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print

Address <http://209.144.148.115/ek/helpyoucanuse.nsf/registergold> Links

Visitor Registration

Name:

Phone:

☒ [Feel free](#) to sell my name to mailing lists to exploit me and profit your company
[More info](#) on releasing your name

Address:

City / State / Zip:

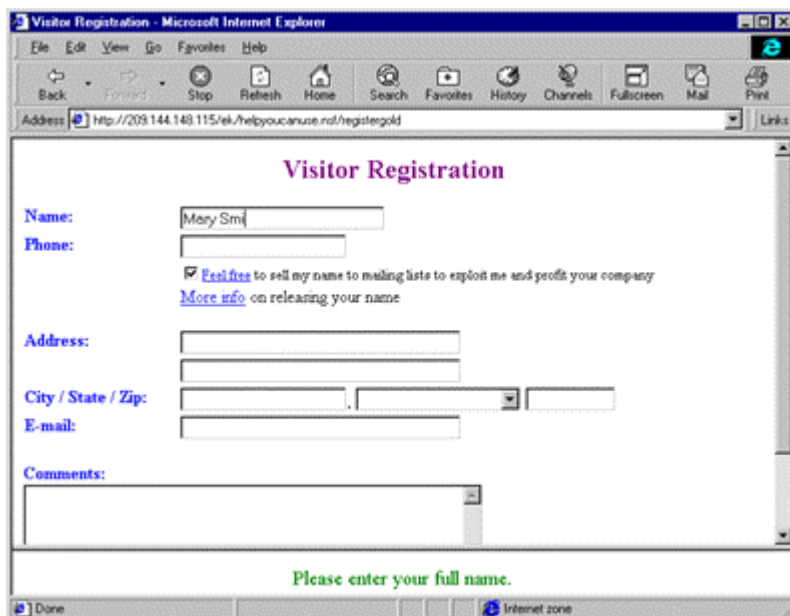
E-mail:

Comments:

Click in any field for help on that field.

Opening page <http://209.144.148.115/ek/> Internet zone

When you put your cursor in a field, the red "Click in any field for help" text changes to a green help description for the field you're in. For example, the following screen shows how, when my cursor is in the Name field, the help text "Please enter your full name" displays in the bottom frame:



Visitor Registration - Microsoft Internet Explorer

File Edit View Go Favorites Help

Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print

Address <http://209.144.148.115/ek/helpyoucanuse.nsf/registergold> Links

Visitor Registration

Name:

Phone:

☒ [Feel free](#) to sell my name to mailing lists to exploit me and profit your company
[More info](#) on releasing your name

Address:

City / State / Zip:

E-mail:

Comments:

Please enter your full name.

Done Internet zone

As you tab into each field, the help text at the bottom changes. It displays as long as you're in the field, and disappears when you enter a field with no help text. Although the bottom portion of the window is a separate frame,

notice that the URL for the page doesn't look like a typical framed page. In most framed applications the URL never changes -- it takes you back to where the frameset was first launched. But this URL takes you directly to the form, so the user can bookmark the page or forward the URL to a friend. This feature has nothing to do with JavaScript, but it is an important usability enhancement to the field help solution.

If you tab into the Phone field, you'll notice that in addition to the line of green text in the help frame, you see a link labeled *More Info*. A simple JavaScript function executes when the field gets focus, and writes the help text out to the help frame:

Visitor Registration - Microsoft Internet Explorer

Address: http://209.144.148.115/ek/helpyoucanuse.nsf/registergold

Visitor Registration

Name: Mary Smith

Phone: (703)

☒ Feel free to sell my name to mailing lists to exploit me and profit your company

[More info on releasing your name](#)

Address:

City / State / Zip:

E-mail:

Comments:

Area code first, like this: (317) 555-1212. -- [More Info](#)

Clicking on *More Info* brings up a separate pop-up window that contains a further help description. This pop-up window displays using only JavaScript on the browser -- the server is not accessed. This is trivial if your application is running on a fast intranet, but very nice if you have users accessing your site with modems or through overloaded proxy servers!

Visitor Registration - Microsoft Internet Explorer

Address: http://209.144.148.115/ek/helpyoucanuse.nsf/registergold

Visitor

Name: Mary Smith

Phone: (703)

☒ Feel free to sell my name to mailing lists to exploit me and profit your company

[More info on releasing your name](#)

Address:

City / State / Zip:

E-mail:

Comments:

Area code first, like this: (317) 555-1212. -- [More Info](#)

Help

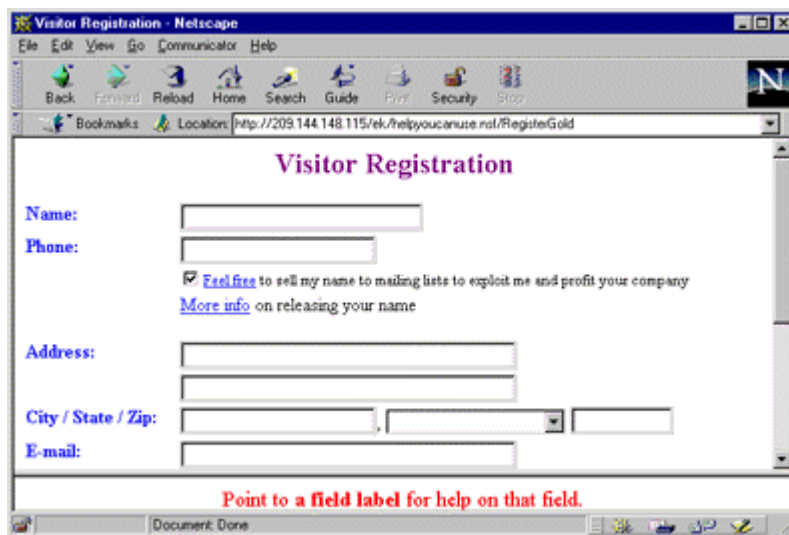
Please include **country code** (if outside the United States) and an **extension** if applicable.

close

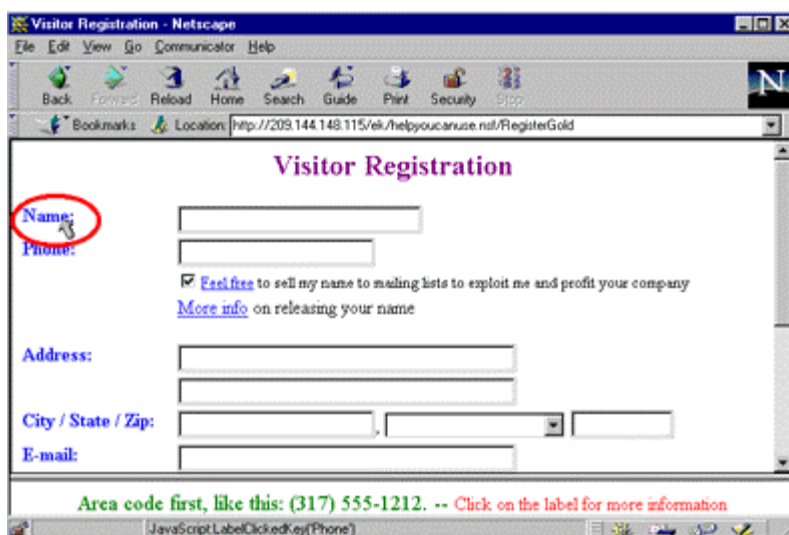
Using the techniques in Netscape 4.x

The field help techniques demonstrated above work great for Netscape 3.x as well as both Internet Explorer 3.x and 4.x browsers. However, there is a problem in at least some of the Netscape 4.x browsers when a field has JavaScript code associated with its events. Namely, the Tab key no longer takes you to the next field, but instead takes you to the browser's URL area. You can try this if you use the Register form with a Netscape 4.x browser. You'll find that if you enter data into the first field (Name) and then press the Tab key, you'll wind up in the URL line. If you press Tab again, you'll go to the second field (Phone). It's usable once you know what it's doing, because you can just press Tab twice to go from field-to-field. But it certainly doesn't help make your form easier to use.

We haven't found a way to prevent the tabbing problem directly, but we have found an alternative way of displaying the field help that avoids this bug. Instead of seeing the field help when you tab into a field, the alternate technique displays it when you point to (mouse over) the field label. With this technique, our default help frame instruction doesn't direct the users to enter a field for help -- instead, it directs them to "point to" a field label for help on that field. This appears with a Netscape 4.x browser in the following screen:



When you point to a field label, you see the field help display in the frame, as shown here (the mouse is over the Name field):



So, all you have to do is point to a field label to see the help text. And if there is more text available in a pop-up window, as in this example, you see the instruction to click on the field label for more information. Clicking on the field label pops up the same help text window we saw in the first example.

The mouse-over field help works well for Netscape 4.x. In fact, you can also use this approach for Internet Explorer 4.x, if you prefer it. However, this technique doesn't work for Netscape 3.x or Internet Explorer 3.x, because it uses an HTML construct these earlier browsers don't support: unformatted anchor tags, which make the field labels "hot" without having them underlined.

The RegisterGold form in the sample database handles both the field focus solution and the mouse-over solution. It detects the browser type, and uses the field focus solution for browsers other than Netscape 4.x, for which it uses the mouse-over solution. You can also override the settings in a single place at the top of the form if you want to use the mouse-over solution for Internet Explorer as well. (Part 2 in this series describes the details of the mouse-over technique.)

How it all works

The remainder of this article describes the techniques for creating the help frame, field help, and pop-up help shown above. We'll cover the basic components of the field focus solution. Then, Part 2 in this series describes the alternate technique used in the Netscape 4.x "mouse-over" solution, and explains how to combine the techniques, detect the browser, and dynamically generate the appropriate JavaScript.

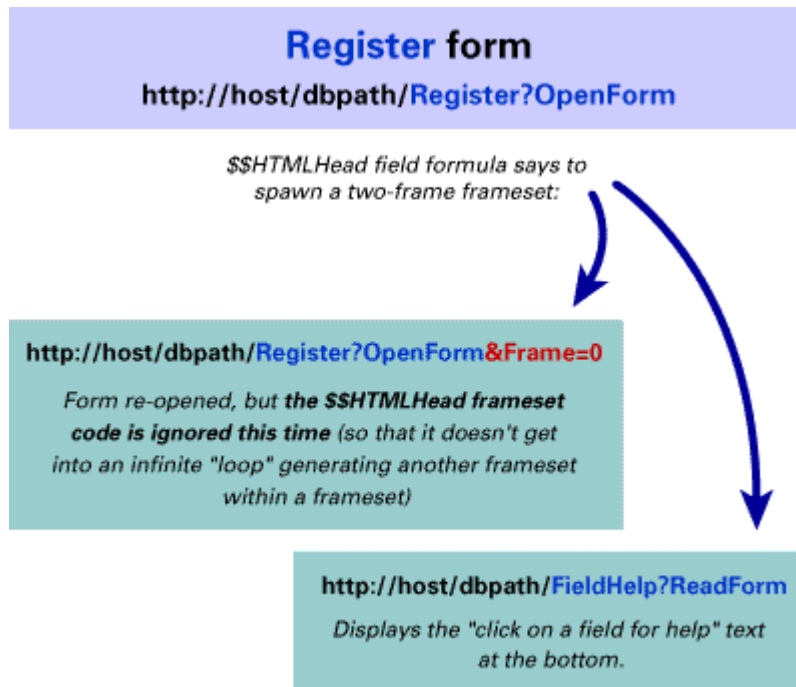
Creating no-hassle help frames

Let's start with the no-hassle help frames. Many users complain about frames, because the URL they see at the top of their browser doesn't change as they navigate through the site; consequently, it's difficult for them to see where they are, to pass a page URL along to someone else, or to effectively bookmark a page (bookmarking a frame normally doesn't return you to that page within the frameset). In this solution we provide the benefits of frames -- context-sensitive help text that doesn't scroll off the screen -- without the hassle, because users can always copy or bookmark the URL shown at the top of their browser for an easy return to the form.

One way to accomplish this is to have an extra form for each form in your database that does nothing more than define the frameset. However, it is a lot of extra design overhead to have a separate frame form for each form in your application. Also, you've got to be careful how you access that form throughout your application, especially if you're designing for both Notes and Web users. (For example, if you had a separate CustomerSetup form to define the frameset for your Customer form, your URLs would need to end in *CustomerSetup?OpenForm*, while your Notes clients would need to access the Customer form.)

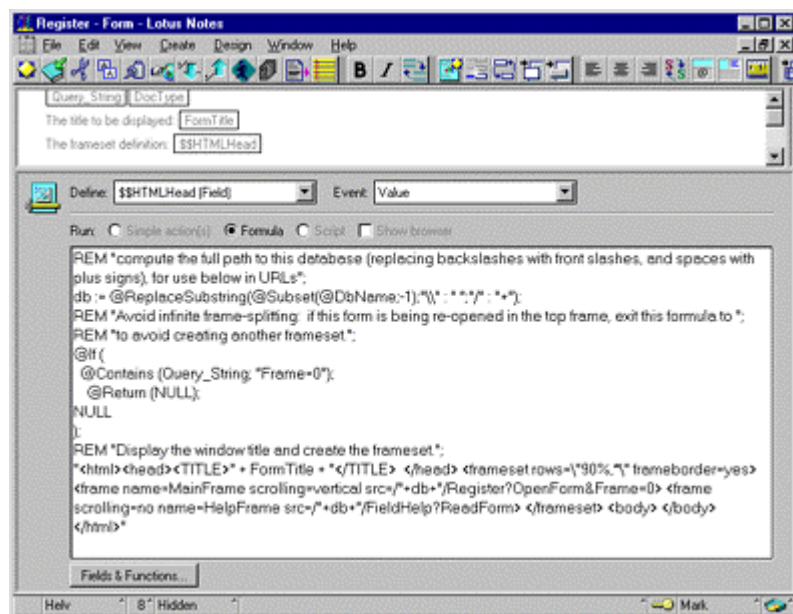
An alternative approach, which we will describe here, lets you define the frameset directly within your form, rather than defining a separate frameset form. We use an `$$HTMLHead` field on the form to direct the browser to set up a frameset, opening (reopening, actually) your form in the top frame, along with a field help form in the bottom frame.

The following diagram illustrates what happens when users create a document with the Register form, using the URL ending in `/Register?OpenForm`:



In the sample database, the Register form contains the working example we will use to illustrate the no-hassle frames and the other portions of the field help solution. (**Note:** The RegisterGold form is a more robust solution, handling problems with tabbing in Netscape 4.x and other issues. However, we will first describe the basic techniques by using the simpler Register form.)

Here is the Register form shown in design mode, with the `$$HTMLHead` formula showing:



When users click on a link to register, the URL ends in `Register?OpenForm`. This URL causes Domino to generate the HTML and send it back to the browser, just like it would any other page. But since the page it sends back has a

frameset definition in its <Head> tag, the browser ignores everything below the </Head> tag and instead goes back to the server to get the URLs defined in that frameset. Those pages then display in the frames.

The unusual thing about this frameset is that the URL for the top frame uses the *same form*, Register, as the initial URL that triggered the frameset. As described above, we would normally use a separate form to define the frameset, and it would launch a form such as Register in the top frame. But in this case, we use the same form to launch the frameset and display the main form in it; the form's frameset definition directs the server to re-open itself. So, the HTML that defines the fields on the Register form (Name, Phone, Address, and so on) is actually sent to the browser twice. The first time, the browser ignores those fields because it stops processing the HTML for the page once it encounters the frameset definition (before it runs into the field definitions). The second time -- when the same form is re-opened in the top frame of the frameset -- the browser displays the fields Name, Phone, Address, and so on, in the top frame.

Sending the same form HTML to the browser twice is also the deficiency with this approach: the HTML is generated and sent to the browser twice, which could slow the page loading if it's a long form. Of course, you could use computed subforms to avoid generating the main form fields twice. You would simply define the "meat" of the form (Name, Phone, and so on) on a subform, and include a computed subform on the Register form that uses the "meat" subform only when the form is opened in the top frame.

So, how does the "second opening" of the form skip the frameset stuff again, avoiding an infinite form re-open loop? Look closely at the \$\$HTMLHead formula, shown below. The URL of the first frame, which is the main frame containing the entry fields, calls for opening the Register form *again*, but includes the parameter &Frame=0 (passed on the URL after the ?OpenForm) which tells the "second opening" of that form to skip the frameset.

If you look at the Register form in the sample database, you'll find this formula in the \$\$HTMLHead field:

```
REM "compute the full path to this database (replacing backslashes with front slashes, and spaces with plus signs),
for use below in URLs";
db := @ReplaceSubstring(@Subset(@DbName;-1);"\" : " \"; "/" : "+");
rem "Avoid infinite frame-splitting: if this form is being re-opened in the top frame, exit this formula";
rem "to avoid creating another frameset.";
@if (
  @Contains (Query_String; "Frame=0");
  @Return (NULL);
  NULL
);
rem "Display the window title and create the frameset.";
"<html><head><TITLE> + FormTitle + "</TITLE> </head> <frameset rows=\\"90%,*\" frameborder=yes> <frame
name=MainFrame scrolling=vertical src=/"db+/"Register?OpenForm&Frame=0> <frame scrolling=no
name=HelpFrame src=/"db+/"Blank?ReadForm> </frameset> <body> </body> </html>"
```

Looking at the above formula, skim down to the <frameset> tag. The code is creating a frameset with two frames. Notice the URL for the first frame contains the text *Register?OpenForm&Frame=0*. We want the top frame to be the Register form, but we don't want that form in the top frame to again split into two frames and try to re-open itself again. So we pass the &Frame=0 parameter (we invented this parameter; you can pass anything you like after the &) on the URL, which tells our form to just open normally in the top frame, without creating another frameset.

So, looking back up to the top of the formula, the code @If (@Contains (Query_String; "Frame=0")) is simply checking for that &Frame=0 parameter and exiting the \$\$HTMLHead formula if it finds it.

Note: We have hard-coded the form name into this formula. We also have not accounted for a URL that opens an existing Register document in edit mode -- a URL that would end in ?EditDocument instead of Register?OpenForm. The RegisterGold form has an \$\$HTMLHead field that handles those situations, so you can use that field without changing the formula at all.

The second frame is the help frame, which is opened with the FieldHelp form (it starts out without any field help). The FieldHelp form simply contains static text to instruct the user to enter a field to see context-sensitive help, as follows:

Please fill out the form. Click in any field for help on that field.

Getting out of the frameset when the form is submitted

Since we have put the Register form within a frameset, we need to make sure that when users click Submit, the returned page comes back outside of the frameset. Otherwise, they will see the return page (for example, "Thank you for submitting your registration") come back in the main frame, with the help frame still in place at the bottom.

We can avoid this with the following JavaScript (from the top of the in-line JavaScript on the Register form):

```
var f = document.forms[0];  
// Set the target of actions on the form to open the next page outside the frames (we don't want the help  
// frame) showing once the user presses the 'Submit' button  
f.target='_top'; // If you are launching your form within another navigational frameset, change this to one of your  
frames.
```

The first line creates a global variable (meaning it's available to any function defined in this form) pointing to the first form on this document. Remember, these are JavaScript objects, not Notes objects; the document is the contents of the window -- what we commonly refer to as the page -- and each document can have one or more forms (HTML forms) defined on it. These forms can all be referenced from the document object in the *document.forms* array, and the first one (we usually have only one) is *forms[0]*, because all JavaScript arrays start with position zero.

We then set the target property of that form to *'_top'*, which is a reserved name for a frame, meaning "not in a frame." We could have done this in one line of code -- rather than using the temporary variable *f*, we could have simply used the following:

```
document.forms[0].target = '_top';
```

But by creating the variable *f* first, we can reference *f* in any other JavaScript we use on the Register form.

Creating basic field help

Now, let's look at how to create the basic field help. If you look on the Register form (in Designer mode) at the HTML attributes for the FullName field, you'll see this formula:

```
msg := "Please enter your full name here.";  
popup := NULL;  
"size=25 onFocus='\FrameHelpText(\"" + msg + "\", \"\" + popup + "\")\""
```

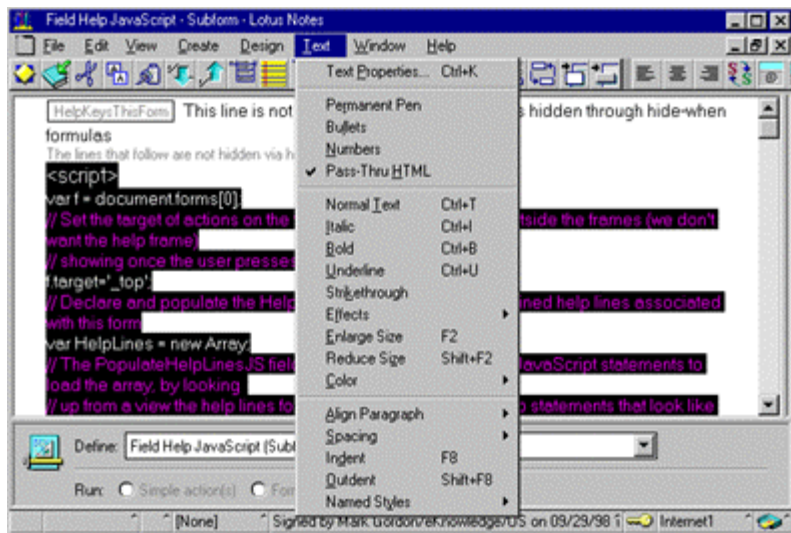
The extra quotation marks required to delimit this text within the Notes formula window make the syntax look confusing, and the fact that we've first set the help text and pop-up text (of which there is none defined for this field) to the temporary variables *msg* and *popup* makes it harder to read. But using the temporary variables makes it easier to use; you can simply copy and paste the formula into different Domino fields and then change what gets assigned to *msg* and *popup* in each one.

The code assigns a call to the JavaScript *FrameHelpText* function to the *onFocus* event of the field. You can assign any HTML field an *onFocus* event. If you defined this field using pass-thru HTML rather than creating a Domino field, or if you simply view the HTML source of the Web page when it is running, you would see the native HTML syntax, without the extra Notes formula quotes:

```
<INPUT NAME="FullName" size=25 onFocus='FrameHelpText("Please enter your full name here.", "")'>
```

The *FrameHelpText* function is defined directly on the Register form, below the fields. The JavaScript on the Domino form (or in any native HTML file) must begin with the HTML tag *<script>* and end with *</script>*. Between the script tags, you can write regular JavaScript code (in-line script) as well as define functions and subroutines, which don't execute until they're either called from the in-line script or from events such as the *onFocus* event we coded above.

Like regular HTML that you put on a Domino form, you need to specify the code as pass-thru HTML by choosing Text - Pass-Thru HTML:



Here is the relevant portion of the JavaScript code for the FrameHelpText function (the portion not shown here handles the pop-up window, which we will discuss later):

```
function FrameHelpText(msg, popupText) {
  // Write the help text to the help frame
  window.parent.HelpFrame.document.write("<font size=4 color=green><Center>" + msg);
  ...
  ...
  window.parent.HelpFrame.document.close(); // this closes out the document (allowing the text to display), not the
  frame or window
}
```

If you have worked in an object-based language like C++, Java, or LotusScript, you should find JavaScript pretty easy to follow if you first understand its object model. It is, of course, beyond the scope of this article to fully explain that object model, but we will discuss those portions which relate to the code we show here.

We want to write the help message out to the help frame, but our JavaScript that does the writing is in a different frame (the main frame, where the user is clicking on fields). So, how do we tell the browser to write a message to a different frame? We "navigate" through the object hierarchy. The Help Frame is an object that has the same parent as the main frame -- the document that defined both frames. So, *window.parent.HelpFrame* takes us from the current frame where the JavaScript is, up to the parent frame, and then down to the frameset's other frame, the help frame.

The code *window.parent.HelpFrame.document.write* does two things: it navigates to the help frame and then writes out the text. It doesn't write the text directly to the frame object, though, because the frame object doesn't have a write method. Instead, it writes the text to the document in that frame. Both frames and windows can have document objects associated with them. Don't confuse a document in the browser object model with a Notes *document*. The browser *document* is simply the Web page. The code *window.parent.HelpFrame.document.write* means to write text to the document within the help frame, which has as its parent the top window.

Clearing the help frame

When the user enters a field for which we have no field help defined, we want to clear the help frame to avoid confusion. The formula in the HTML attributes for the field looks like this (see the Address1 and Address2 fields on the Register form for examples):

```
onfocus='\ClearHelpFrame()\'
```

The ClearHelpFrame function looks like this:

```
// This function clears the help frame. Intended to be called from fields that have no help text.
function ClearHelpFrame() {
    window.parent.HelpFrame.document.write ("");
    window.parent.HelpFrame.document.close();
}
```

Creating pop-up help

Next, we can look at how to create the *More Info* link that launches a dynamic pop-up help window. In the HTML attributes for the FullName field, we had a call to the FrameHelpText function that left the second argument blank. We didn't want any pop-up help for FullName, just the single line of help text, but the State field includes the second, *popup* argument, as shown here:

```
msg := "Canadian provinces are below the states.";
popup := "Obviously this sample application does not handle addresses <b>outside</b> the United States.";
"onFocus=\FrameHelpText(\"" + msg + "\", \"\" + popup + "\")\""
```

Let's take another look at the FrameHelpText function, which this time includes the lines we left out before (shown in bold):

```
function FrameHelpText(msg, popupText) {
    // Write the help text to the help frame
    window.parent.HelpFrame.document.write("<font size=4 color=green><Center>" + msg);
    if (popupText == "") {
        // Close out the formatting tags since there is no More Info link to be shown
        window.parent.HelpFrame.document.write("</font></Center>");
    } else {
        // Show the More Info link
        window.parent.HelpFrame.document.write (" -- ");
        window.parent.HelpFrame.document.write("<A HREF='JavaScript:top.MainFrame.showHelp(\"" +
        popupText + "\")'>");
        window.parent.HelpFrame.document.write("More Info</A></Center></font>");
    }
    window.parent.HelpFrame.document.close(); // this closes out the document (allowing the text to display), not the
    frame or window
}
```

The If statement checks to see if the *popupText* argument is blank, and if it is, there is no *More Info* link to display. Otherwise, the link is written, which includes the *HREF* JavaScript code instead of a URL. The code is simply a call to the function we've defined called *showHelp*, passing as an argument the help text to be displayed in the pop-up window. (Notice the double equals signs, ==, in the If statement. In JavaScript, a single equals sign is for assigning values and a double equals sign is for comparisons. Leaving that second equals sign out may be the novice JavaScript developer's most common mistake! Some browsers warn you, but others let you search forever for flaws in your logic before you realize your error.)

The showHelp function, shown below, uses the *window.open* method to create a pop-up window. The first argument to *window.open* is the URL of the page to load into the window. If you leave this argument blank, like we have here, the window will pop up empty without a hit to the server. Then, you can use *document.write* to write the HTML directly onto that window. It may not take long for a pop-up window to load from the server over a fast intranet, but if the pop-up help is only for additional information, users may avoid the *More Info* link if they find it takes more than a couple of seconds to display.

Here is the showHelp function:

```
var winHandle; // this variable is defined before the function call, so that it is a global variable available to other functions
// This function pops up the help window with the more in-depth help text for that help topic
function showHelp(popupText) {
    winHandle = window.open("", 'helpWin', 'height=300,width=200');
    winHandle.document.write("<h1>DocHelp</h1>");
    winHandle.document.write(popupText);
    winHandle.document.write("<form><input type=button value=close onclick=\\\"closeWin()\\\"></form>");
    winHandle.document.write("<script>function closeWin() { window.close(); } </s> + \"cript>");
    // Put the focus on this window. That way, if the window was already open from another help popup,
    // it will be brought to the front.
    winHandle.focus(); // IE3 does not support this line The Gold version of this function on the gold JS subform handles this.
    winHandle.document.close();
}
```

The function writes out to the pop-up window the text passed in via the *popupText* argument. It also defines a *<form>* for the sole purpose of showing a Submit button. Because there is no *METHOD=post ACTION=urلتocreatedocument* included in this form, submitting it doesn't send any data (you may not be familiar with the *METHOD=post ACTION* construct if you've never written native HTML forms; Domino normally generates the code when it sends a form to the browser).

Take a close look at the line of code that writes the JavaScript function to close the pop-up window:

```
winHandle.document.write("<script>function closeWin() { window.close(); } </s> + \"cript>");
```

We have separated the *<script>* tag by ending and then starting the string again so that the JavaScript interpreter on the main window doesn't mistake it for the end of the *<script>* tag on that main window. The JavaScript interpreter on the pop-up window will correctly identify that tag as applying to the script on that window.

Conclusion

So far, you've seen the basics for how to add context-sensitive field to your Domino applications. We've covered techniques for creating a separate help frame, basic field help, and pop-up help. Next time, we'll get into the details of how you can create reusable help topics, and create field help that appears for Netscape 4.x users when they mouse over a field label. You'll also see how Domino makes it easy for you to create field help, because it can generate the code for you. See you next time!

ABOUT MICAH

Micah Blalock is an Internet consultant and instructor at WorkFlow Designs, Inc. in Dallas, Texas. Blalock has wide experience developing solutions for the commercial database, medical and PC game industries. Currently, his focus is on development and technical training with specialization in application development for Internet technologies such as Lotus Notes/Domino and JavaScript. Blalock also lends his combination of technical and training expertise in developing the technique-oriented courseware for Internet development offered through WorkFlow Designs TopGun Academy curriculum.

Copyright 1998 Iris Associates, Inc. all rights reserved.

Quick steps to adding field help to your applications (sidebar)

If you want to quickly implement field help in your own application -- regardless of whether you've used JavaScript before or even read this article -- simply download the sample database and follow these simple instructions. All you will need to do is copy a few design elements from the sample database into your application, and then paste a formula into each field for which you want help, changing the help text you want displayed. If you want reusable field help, there is an additional form and a few elements that you will need to copy into your application. You will also need to compose the field help documents before you can reference them in your forms. Here are the detailed steps:

1. Copy the following design elements from the sample database into your database:

- The "Field Help JavaScript Gold" subform
- The "Field Help Gold" form

2. To set up your form for field help, do the following:

- Open the RegisterGold form in the sample database, and copy everything between the green horizontal bars onto your form.
- Edit the FormTitle field to reflect the title of your form.
- Copy everything between the "Pull in help topics" line through the <Computed Subform> formula. Paste it onto your form.
- If you are *not* planning to create any reusable help topics, but want to simply define the help text as needed on a field-by-field basis, delete the FormHelpKey field. Then, skip to Step 5.

The screenshot shows the 'RegisterGold - Form - Lotus Notes' window. The menu bar includes File, Edit, View, Create, Design, Text, Window, and Help. The toolbar contains various icons for editing and formatting. The form content includes:

- A label 'emailLabel' and a text field 'Email'.
- A red text annotation: 'This field references a reusable help key.'
- A 'Comments:' section with a 'Body' text field.
- A red-bordered box containing the following text:
 - 'Pull in help topics with this form help key: FormHelpKey'
 - 'Include *Field Help JavaScript Gold* subform only in edit mode:'
 - '<Computed Subform>'
- A '\$\$Return' button at the bottom.
- A status bar at the bottom showing 'Helv', a font size of '10', and '[None]'.

3. To set up reusable help, do the following:

- Copy the Help Topic form and the Help Topics view to your database.
- Compose a help topic for each reusable help item.
 - Make up a help key for each one; you will reference this help key in any fields for which you use that help item.
 - Specify the form or forms on which you want to use that help item.
 - Specify help text, and, if you like, pop-up help text. Use HTML to format the pop-up text (see the Email help topic document in the sample database).

- In the screen above, notice the FormHelpKey field. You should have this field on all the forms on which you will use the field help solution. Specify the form name you identified in your help topic. This is what tells your form to pull in the appropriate help keys that are used on it.
4. For each field for which you want to include a **reusable help** topic, copy and paste the field label and field formulas into your form, as follows:

- Copy the phoneLabel field from RegisterGold and **paste it over** (replacing) your static text. This is for Netscape 4.x browsers, and defines the help text to show when users point to the field label. Change the first two lines of the formula to specify your label text and the help key (in the phone example, both the field label and the help key are called *Phone*):

```
LabelText := "Phone:";
HelpKey := "Phone";
```

- Copy the HTMLAttributes formula from the Phone field from RegisterGold and **paste it into** the HTML Attributes for your field. This is for all browsers except Netscape 4.x, and defines the help text to display when the field gets focus. Change the first line of the formula to specify the same help key you specified above. Also, if you had already defined HTML attributes, such as the size of the field, put them into the third line (otherwise, set that line to NULL). Here is what the formula looks like, with the things you should change shown in bold:

```
HelpKey := "Phone";
type := "FieldFocus";
otherHTML := "size=20";
@if (
  FieldFocusHelp;
  OtherHTML + " onFocus='FrameHelpKey(\"" + HelpKey + "\", \"" + type + "\")\"";
  OtherHTML
)
```

5. For each field for which you want to **define the help text on the fly**, copy and paste the field label and field formulas into your form, as follows:

- Copy the nameLabel field from RegisterGold and **paste it over** (replacing) your static text. This is for Netscape 4.x browsers, and defines the help text to show when the users point to the field label. Change the first three lines of the formula to specify your label text, the help text and the pop-up help text. If you don't want pop-up text for a field, set *popup* equal to NULL. Here is what the first three lines of the formula look like, with the things you should change shown in **bold**:

```
LabelText := "Name";
msg := "Please enter your full name.";
popup := "Please do not use nicknames, but rather your full name as printed on your Lotus Notes Application Development 1 course completion certificate.";
```

- Copy the HTMLAttributes formula from the Name field from RegisterGold and **paste it into** the HTML Attributes for your field. This is for all browsers except Netscape 4.x, and defines the help text to display when the field gets focus. Change the two lines of the formula to specify the same help text you specified above. Also, if you had already defined HTML attributes, such as the size of the field, put them into the fourth line (otherwise, set that line to NULL). Here is what the formula looks like, with the things you should change shown in **bold**:

```
msg := "Please enter your first name.";
popup := "Please do not use nicknames, but rather your full name as printed on your Lotus Notes Application Development 1 course completion certificate.";
type := "FieldFocus";
```

```
otherHTML := "size=25";  
@If (  
  FieldFocusHelp;  
  OtherHTML + " onFocus='FrameHelpText(\"" + msg + "\", \"" + popup + "\", \"" + type + "\")\"";  
  OtherHTML  
)
```

That's it! You should now have working context-sensitive field help for your Web users!

Copyright 1998 Iris Associates, Inc. all rights reserved.