Notes.net

Iris Today

by
Michael
Patrick

Level: Intermediate
Works with: Designer 5.0
Updated: 10/02/2000

If the phrase "Domino-only e-commerce site" strikes you as an oxymoron, think again. The fact is, amidst all the justifiable excitement surrounding Domino's ongoing integration with platforms such as WebSphere and various relational databases, it's all but forgotten that building a successful e-commerce presence on the Web doesn't necessarily require an array of technologies and products, especially when Domino is part of the mix.

Indeed, there are Notes/Domino customers with a desire to leverage their existing investment and capitalize on Domino's ease-of-use in bringing their product catalogs to the Web. In many instances, these organizations serve a very specific group of customers and may not require a product like WebSphere, nor are their operations elaborate enough to justify the complexity that integrating a relational back-end into their Domino environment would introduce.

One such organization is Liberty Fund, Inc., an educational foundation that, among other functions, offers historical texts on economics, political thought, and so on for sale to the public. When it came time to establish sales on the Web, Liberty Fund saw Domino as both a self-contained platform capable of delivering their e-commerce solution and an intuitive means by which their catalog and customer information could be maintained.

This article is the first in a three-part series that examines some of the underlying techniques used to transform Liberty Fund's catalog into a true e-commerce experience for its customers. In this first installment, I'll focus on site navigation, a challenge faced by all e-commerce applications; how do you get your customers to the catalog items that interest them? Subsequent articles will examine the topics of session tracking, catalog entries, availability notifications, and shopping cart/checkout mechanisms.

The sample databases that accompany the articles in this series contain Liberty Fund's catalog as well as code used to deliver their e-commerce solution. (You can download the sample database for this article, **Liberty Fund Library 1**, from the Iris Sandbox.) The first sample database is limited to the topics covered in the current article. As the series progresses and successive topics build on one another, additional design elements will be added to the sample database, thus making the "site" more complete. At the conclusion of the series, you will have a set of tools that can be incorporated into e-commerce solutions of your own quickly and with relative ease.

This article assumes experience designing Notes/Domino applications using Domino Designer R5.

Navigational challenges

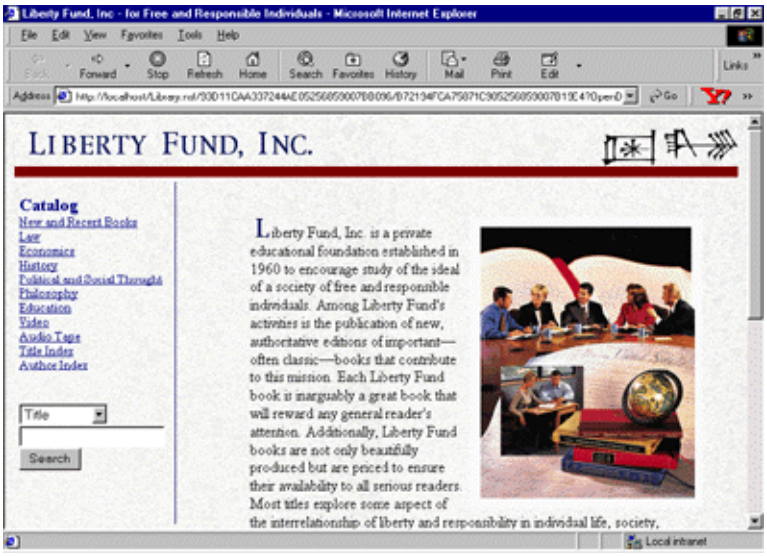
Nowhere should the axiom "form follows function" hold more true than a Web site designed to sell products. No amount of pizazz will compensate for a site that is not intuitive or hinders the browsing and buying process. One of the single biggest challenges in constructing a simple-to-use e-commerce site is providing basic and consistent navigational elements throughout the site. As users move from page to page, they should be met with items they immediately recognize, allowing their attention to be directed toward page *content,* where the potential revenue resides.

© Copyright 2000 Iris Associates, Inc.

To that end, we'll start our e-commerce site exploration at the highest level: how to ensure that your customers can get to where they want to go—without pulling their hair out while doing so.

You can see our first two topics—searching and dynamic navigation—in action by previewing the Liberty Fund home page in your browser. To see the home page:

1. Open the sample database, **Liberty Fund Library 1** (Library.nsf), in Notes.
2. Switch to the Pages view.
3. Open the first document, Liberty Fund, Inc. - for Free and Responsible Individuals.
4. Choose Actions - Preview in Web Browser.



Searching

For an e-commerce site to be at all successful, it must offer an effective search mechanism. When customers cannot locate products quickly and easily, it won't take them long to find one of your competitor's sites where they can. Fortunately, Domino's native full-text search capability fulfills this requirement quite handily. For a primer on searching Domino databases on the Web, you might want to review the *Iris Today* article, "**Filtering data for Domino Web users**."

Ideally, the search mechanism should be available anywhere within the site that we choose to include it, keeping the code behind it identical in all instances. Although subforms provide the requisite reusability needed in such a case, an even more flexible option is the use of a shared field. When used to excess, shared fields can negatively impact performance, but they are a powerful tool when used in the following way.

Take a look at the form titled "Page" (from which the Liberty Fund home page is created) in Designer. (To examine the sample database's design, open it in Designer, or with the application open in Notes, choose View - Design.)

© Copyright 2000 Iris Associates, Inc.

Notice the two shared fields (denoted by the heavy border surrounding them) in the left column below the red line. The first, ContentsHTML, builds the navigation links covered in the next example. The second, SearchHTML, contains the code to enable the search capability. If you refer to the Liberty Fund home page, you'll see the results of both fields stretching down the left side of the page.

Let's look at the formula behind the SearchHTML shared field:

```
LibraryDBW := @ReplaceSubstring (@Subset (@DbName; -1); "\\" : " "; "/" :
"+");
"[</FORM><FORM METHOD=post ACTION=\"/" + LibraryDBW +
"/ViewSearchGeneric?CreateDocument\" ENCTYPE=\"multipart/form-data\">"
+
"<BR>" +
"<SELECT NAME=\"SearchType\">
<OPTION VALUE=\"Title\" SELECTED>Title
<OPTION VALUE=\"Body\">Description
<OPTION VALUE=\"Author\">Author
<OPTION VALUE=\"MediaISBNs\">ISBN
<OPTION VALUE=\"All\">All
</SELECT><br>" +
"<INPUT NAME=\"SearchString\" SIZE=\"20\" MAXLENGTH=\"50\"><br>" +
"<INPUT TYPE=Submit Value=\"Search\"></Form>]"
```

The first line is the standard method for determining the path to the current database (and we'll see this same formula throughout the following examples.) I'm also replacing the double backslashes (\\ ) with a single forward slash (/) and spaces with a plus sign (+); otherwise, the links we build that contain these values wouldn't function properly, because URLs don't recognize double backslashes (\\ ) and spaces.

The remainder of the formula places an HTML form on the current page. Why are we doing this? In the case of searches, we want the button used to initiate the search to perform a specific action rather than relying on the page's default behavior—a submit of the entire page. To accomplish this, we first include a closing form tag ("</FORM>") in order to terminate the form that Domino automatically generates. Next, we include the search form. The FORM tag's ACTION property is the crucial part. In effect, it says that when the search form is submitted, a document should be created using the ViewSearchGeneric form. We'll look at this form in a moment.

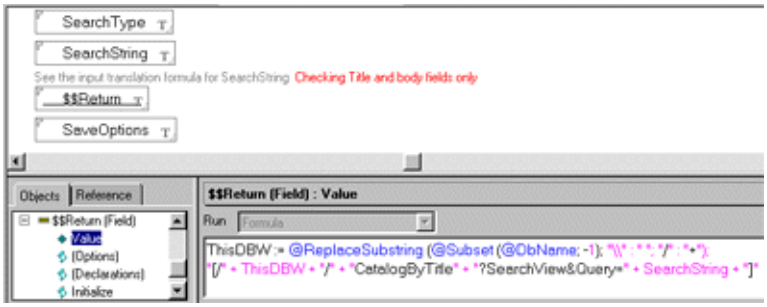
Below that, the SELECT tag places a drop-down list on the form, populating its choices via the OPTION tags. In this example, there are five values from which to pick, with Title serving as the default value. Notice too that the SELECT tag's NAME attribute has been set to SearchType. Finally, the first INPUT tag places a text box on the form with the name SearchString and the

© Copyright 2000 Iris Associates, Inc.

second INPUT tag, whose TYPE is Submit, places a submit button on the form.

If you've downloaded the sample database but have yet to full-text index it, you'll need to do so to see this example in action. Once the database is indexed, conduct a search from anywhere within the site. If you previewed the Liberty Fund home page, simply use the search fields located there. Provided you select a field value to search other than "All," the search will be limited to only that field within the catalog and if any matches are found for the string you supply in the text box, they will be displayed on a results page. For example, try searching by Title for Smith. The search should return five titles.

How does the search work "behind the scenes"? First, remember that a new document is created by the ViewSearchGeneric form when a search is submitted. The sole purpose of this new document is to format and pass the search to Domino. After that, it isn't of any value, so it's not saved thanks to a computed SaveOptions field which is set to 0. You can open the ViewSearchGeneric form and take a look.



The first thing to note is the SearchType and SearchString fields, which are both editable and text. Sound familiar? Both were the names of elements added to our pages by the SearchHTML shared field. Both fields are included on the ViewSearchGeneric form so that their values can be captured when the search is initiated—SearchType storing the field to be searched (Title) and SearchString storing the search value (Smith). Additionally, the SearchString field has the following input translation formula:

```
x := @ReplaceSubstring (SearchString; " "; "+");
@If(SearchType=Null | SearchType="All";x;"[" + SearchType + "]=" + x)
```

Since URL's don't like spaces, the first line takes the SearchString field and replaces all spaces with "+." The second line says that if the customer did not choose a field to search against or they picked All (meaning they want to find the search string anywhere in the catalog) simply return SearchString (Smith). If the customer *has* picked a field, return the field name surrounded by brackets and set equal to the value in SearchString. So, our example from above would end up looking like this:

```
[Title]=Smith
```

This is the shorthand equivalent of Domino's search syntax "FIELD Title CONTAINS Smith."

If you eyed the $$Return field on the form and suspected that it would do the real work, you're right, because that's what actually executes the search against the catalog. It is computed for display and its formula is:

```
ThisDBW := @ReplaceSubstring (@Subset (@DbName; -1); "\\" : " "; "/" :
"+");
"[/" + ThisDBW + "/" + "CatalogByTitle" + "?SearchView&Query=" +
SearchString + "]"
```

© Copyright 2000 Iris Associates, Inc.

We've seen the first line before; it's simply computing the path to the current database. The second line is building a relative URL that will search the CatalogByTitle view using the ?SearchView command. It also passes the SearchString as a parameter; in this example, this looks like &Query=[Title]=Smith. Finally, the entire URL is surrounded by brackets, which tells Domino to redirect the customer's browser to that URL. Domino then receives the URL, executes the search, and displays the results.

The only major piece of the search that remains is the $$SearchTemplateDefault form, which will automatically display search results:

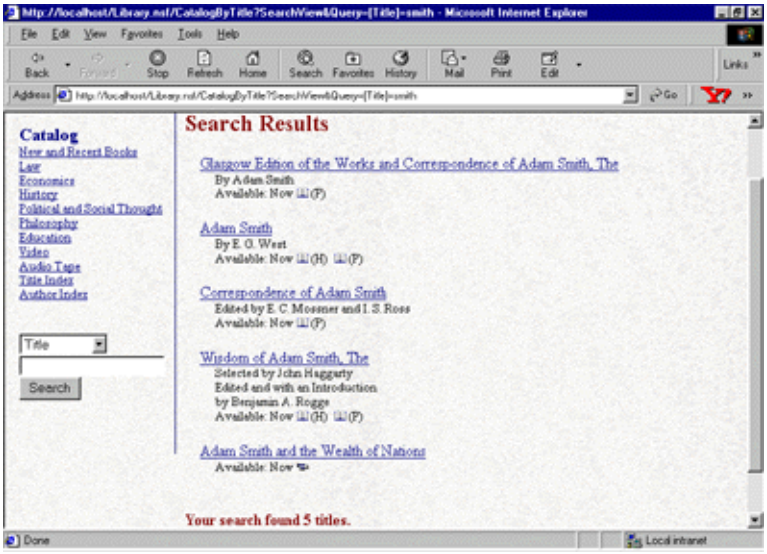
Domino populates the $$ViewBody field with the documents returned by the search as they appear in the view used by the search, CatalogByTitle. The only other feature of interest is something that's new to R5: the Hits and TotalHits fields. Prior to R5, displaying the number of documents returned by a search on a Web page was an ugly affair that involved determining how many links were on the page and then subtracting the number of fixed links that didn't change from search to search. With R5, all you have to do is include computed-for-display fields for either Hits or TotalHits (or both) and Domino populates it with the correct value. Hits returns the number of documents found, and TotalHits returns the total number of instances the search string was found. This sample application is only using Hits, which you'll notice is hidden at the top of the form and then referenced in the computed-for-display field called Count, located immediately below the $$ViewBody field.

The formula for Count is straightforward:

```
"Your search found " + @Text(Hits) + " titles."
```

So, the search by Title for Smith yields the following:

© Copyright 2000 Iris Associates, Inc.



As with the other techniques examined in this article, the search solution is by no means limited to e-commerce solutions; it's applicable to any Domino Web-enabled application where searching is required. When used in an e-commerce setting, your customers will have immediate access to items of interest to them, which is never a bad thing when they're ready to buy your products.

Dynamic navigation links

Allowing customers to search for what they want is only one of the crucial components of an e-commerce solution. Providing customers with an easy way to browse your catalog is just as crucial. One of the easiest methods of offering simple navigation throughout an e-commerce site is to identify a limited number of high-level categories by which products can be grouped and then building links to those categories. You can then place these links wherever they are needed, providing a convenient way to hop across product categories regardless of where the customer currently is on the site.

To see this in action, refer back once again to the Liberty Fund home page. I've already described the search functionality, but above that, notice the links listed under Catalog. These generally represent the broad categories to which the products are assigned.

As mentioned previously, the links are created via the ContentsHTML shared field, which is a text field computed for display. Here is the formula behind the field:

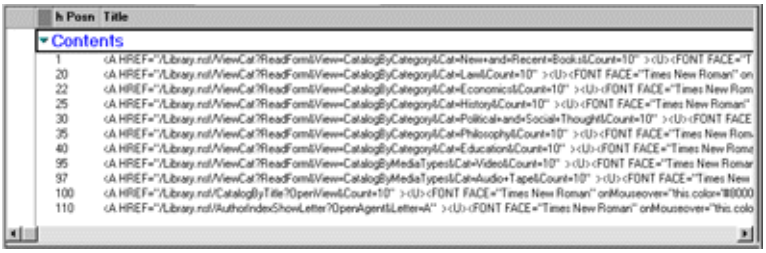
```
ThisDBW := @ReplaceSubstring (@Subset (@DbName; -1); "\\" : " "; "/" :
"+");
x := @DbLookup ("Notes" : "NoCache"; @Subset (@DbName; 1) : ThisDBW;
"(NavigatorEntriesHTML)"; "Contents"; 3);
strContents := @If (@IsError (x); NULL; @Implode (x; NULL));
strContentsHeader := "<BR><B><FONT SIZE=4 COLOR=\"000080\"
FACE=\"Times New Roman\">Catalog</FONT></B><BR>";
"[" + strContentsHeader + strContents + "]"
```

The first line is simply setting a temporary variable equal to the path of the current database. The second line performs a @DbLookup to a hidden view called (NavigatorEntriesHTML). Using a key of Contents, it grabs the third column of the view. You'll see that view in a moment, but let's assume the @DbLookup returns multiple values. The third line then takes the @DbLookup results (assuming the lookup was successful) and implodes them together with a null separator. Line four simply contains the HTML for

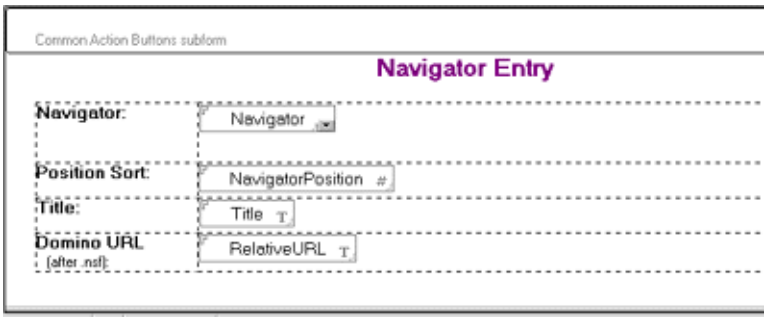
© Copyright 2000 Iris Associates, Inc.

the Catalog heading that you saw at the top of the links. Finally, the last line concatenates the heading and the imploded @DbLookup results, surrounding them with square brackets so that Domino will treat this result as passthru HTML.

Of course the preceding field formula begs the question: why are the links retrieved from a view rather than simply included as part of the formula? They certainly could have been part of the formula, but that would not make for an easily maintained, dynamic site. A central tenant of application design is the allowance for look-and-feel and content changes with as little modification to the actual application code as possible. With this in mind, let's look at the hidden view (NavigatorEntriesHTML) in Designer:



You'll notice that each link is maintained on a separate document in the view. This allows us to add, delete, modify, or rearrange our navigation links in one central location from which the entire site inherits. I'll come back to the view in a moment; let's take a quick look at the Navigator Entries form, which is used to create the documents that will contain the link information:



This form contains four fields: Navigator, a text field that allows us to categorize our links (as will be explained below); NavigatorPosition, a numerical field used to sort the documents in their respective categories; Title, which is text that serves as the link text seen by customers; and lastly, RelativeURL, another text field that will hold the URL of individual link destinations.

Going back to the view, the first column is categorized on the Navigator field. In this example, this field contains Contents across all the documents, but if we wanted to maintain multiple sets of links, we'd simply create additional documents and populate Navigator with another value. If you refer back to the @DbLookup from the ContentsHTML field, you'll see that it's using Contents as the key. So, if we had a different set of links categorized under, for example, General Links, we'd simply use the same @DbLookup, substituting, General Links for Contents.

The second column sorts on the NavigatorPosition field and will control the order in which the link documents appear. The third column represents the @DbLookup's target: the HTML representing the individual links. A single document's value for this column looks like this:

<A

© Copyright 2000 Iris Associates, Inc.

```
HREF="/lf/library.nsf/ViewCat?ReadForm&View=CatalogByMediaTypes&Cat=
Audio+Tape&Count=10"  ><U><FONT FACE="Times New Roman"
onMouseover="this.color='#800000';"
onMouseout="this.color='#000080';">Audio Tape</FONT></U></A><BR>
```

I'll go into the link details a little later in the article, but it's important here to examine just how it was constructed. The formula for the third column is:

```
ThisDBW := @ReplaceSubstring (@Subset (@DbName; -1); "\\" : " "; "/" :
"+");
"<A HREF=\"/" + ThisDBW + "/" + RelativeURL + "\"  ><U><FONT
FACE=\"Times New Roman\" onMouseover=\"this.color=\'#800000\';\"
onMouseout=\"this.color=\'#000080\';\">" + Title + "</FONT></U></A><BR>"
```

By now, only those of you lacking short-term memory are excused from not immediately recognize the first line of the formula. The path of the current database is used in the HREF of the link tag—<A></A>—on the second line. To complete the HREF, the formula then refers to the RelativeURL field on the link documents, which in the case of the HTML link above looks like:

```
ViewCat?ReadForm&View=CatalogByMediaTypes&Cat=Audio+Tape&Count=
10
```

Next, the formula specifies the link's font and its onMouseover and onMouseout properties used to control the link's color as the mouse passes over it. Finally, the link document's Title field represents the actual text displayed to the customer by the link; in the link above, it's Audio Tape.

Putting all these steps together results in set of reusable, dynamic navigation links and will bring a great deal of flexibility to your site. If you wish to do so, refer back to Liberty Fund's home page and examine the links for a moment, hovering over them and examining the URLs they point to in your browser's status bar.

Finally, what was true of the search technique is also true of this one; it is applicable in just about any Domino Web application imaginable. Referring to navigation links from one central location allows for any changes to them to immediately propagate throughout our site, making maintainability and customization of link information/appearance a snap.

Using single-category views

One of R5's great features is the addition of single-category views, which are perfect for displaying categorized product catalogs. As a bonus, they have made life easier on Domino professionals due to their ease of use. And, as with earlier examples, you can take the following example and apply it to just about any Domino Web application imaginable.

In the previous example, I made passing reference to a rather long URL constituting the link to a catalog category. Here's the opening link tag again:

```
<A
HREF="/lf/library.nsf/ViewCat?ReadForm&View=CatalogByMediaTypes&Cat=
Audio+Tape&Count=10">
```

The first thing you should note is that although this URL will result in the display of a category (Audio Tape in this example) in the CatalogByMediaTypes view, the URL itself does not include the ?OpenView command. Instead, it issues a ?ReadForm against the Catalog View Single Category form whose alias is ViewCat. Why display a form instead of opening the view directly? It's a matter of preference, really. First, it would be perfectly acceptable if the URL looked like this:

```
<A
```

© Copyright 2000 Iris Associates, Inc.

```
HREF="/lf/library.nsf/CatalogByMediaTypes?OpenView&Cat=Audio+Tape&C
ount=10">
```

This shorter URL, however, assumes there is a $$ViewTemplate form to handle displaying the view. It's not advisable to make your $$ViewTemplateDefault form contain an embedded single-category view since it won't display non–single-category views correctly. This could have been compensated for by the creation of a "$$ViewTemplate for CatalogByMediaTypes" form. But what if our catalog contained numerous categorized views to be displayed in single-category fashion? Supporting multiple $$ViewTemplate forms can quickly become a burden under such circumstances. The way around this dilemma is demonstrated by the original URL: call a generic form, passing in the names of the view and category to be displayed.

Let's look at the Catalog View Single Category form (remember, the alias, which the URL references, is ViewCat):

As is typical of Web applications, the form includes an editable, hidden field called Query_String that will capture everything in the current page's URL to the right of the question mark (?). This allows us to parse apart the incoming parameters we've included as part of the URL. Working down the form, you'll notice the computed for display Category_d field immediately above the embedded view. This field's formula will evaluate to the name of the category being displayed :

```
@ReplaceSubstring (@Middle (Query_String + "&"; "&Cat="; "&"); "+"; " ")
```

Notice that the formula is grabbing everything returned by Query_String that follows &Cat= and that is delimited by another &. Likewise, the embedded view's Embedded selection property extracts its value from the Query_String. Since a view alias is being passed, it's assumed that the alias in this instance doesn't contain spaces, so an @ReplaceSubstring isn't needed:

```
@Middle (Query_String + "&"; "&View="; "&")
```

And, finally, the embedded view's Show single category property can simply point to Category_d since it already has determined the name of the category to be displayed; so it's formula is just Category_d.

So, if we follow the Audio Tape link used in this example, this is the result:

© Copyright 2000 Iris Associates, Inc.

So, by using a generic form to display a database's multiple single-category views, this example has effectively made use of the idea behind $$ViewTemplate forms without being constrained by their naming convention! Single-category views make a wonderful addition to a Domino developer's toolbox, and they are especially convenient in the context of an e-commerce application. With them, customers can immediately drill into specific subsets of a catalog, and developers don't have to twist into contortions getting them to work.

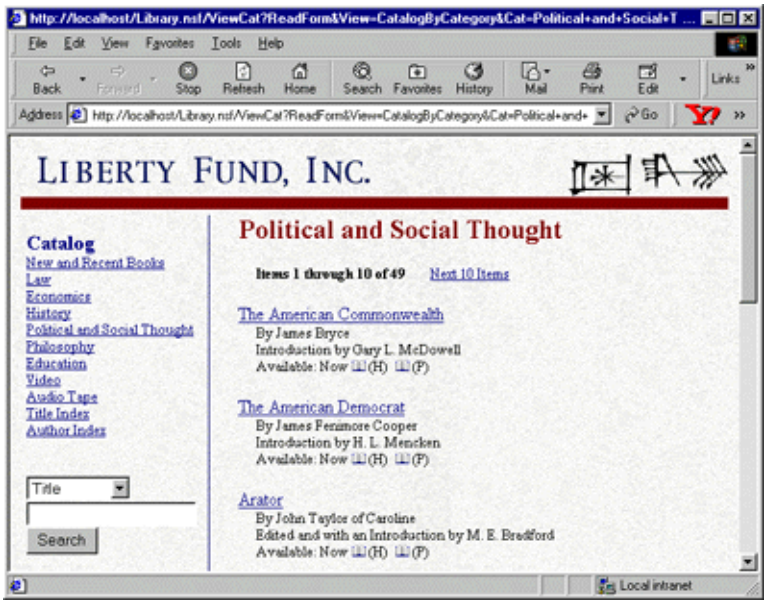
Category navigation agent

Liberty Fund wanted the means to browse catalog categories (some of which are quite large) that avoided swamping customers with a single page containing an unmanageable number of items. They also wanted a way to convey to customers their current position within the category. To that end, a WebQueryOpen agent was incorporated into the forms that display views. This agent calculates the category sequence numbers of the items being displayed as well as seeing to the inclusion of Previous and Next navigation links as needed.

A word of caution: If you choose to add this functionality to your own Domino applications, be aware that running an agent each time a user changes their position within a view will add a certain amount of overhead to the application and the server. For a heavily-used site, such an agent is inadvisable. But for sites accommodating a small number of simultaneous users, you might find that this agent provides a more intuitive way for customers to make their way through the contents of your catalog.

Let's first get a glimpse of what the agent actually does. This is what a customer sees at the start of the Political and Social Thought category:

© Copyright 2000 Iris Associates, Inc.



Notice under the category name it says, "Items 1 through 10 of 49" followed by a link labeled, "Next 10 Items." Take a look at what a customer sees toward the end of the category:

Now there are two links, one for Previous 10 Items and another for Last 9 Items. This demonstrates that the navigation is indeed dynamic since links are only included when needed; at the beginning of a category there aren't any previous items, so a link to that effect isn't included. Similarly, the Next link changes to a Last link near the end of a category and will disappear altogether when the last item is displayed.

The Single-Category Views example that preceded this topic examined the Catalog View Single Category form, and we'll consider it again here. The form's WebQueryOpen object contains the following formula:

```
@Command([ToolsRunMacro]; "(ViewCat WebQueryOpen)")
```

© Copyright 2000 Iris Associates, Inc.

As the term WebQueryOpen implies, each time this form is called upon by a browser request, it will first execute the specified agent whose purpose is to act upon the page about to be displayed. If you open the sample database in Designer, you'll notice that one of the agents listed is (ViewCat WebQueryOpen). The agent name is surrounded by parentheses because it's a hidden agent, meaning its "When should this agent run?" property is set to "Manually From Agent List." This is of course the agent that performs all the calculations and builds the links.

How does the agent get the information it needs to do its job? Take a look at an underlying URL from a Next 10 Items link:

http://localhost/Library.nsf/ViewCat?ReadForm&View=Catalog ByCategory&Cat=Political+and+Social+Thought&Start=21&Count=10

I've already discussed passing the view and category names as parameters on the URL, both of which are used by the agent. The last two parameters, Start and Count serve dual purposes. First, since both are parameters recognized by Domino, they tell the server which and how many documents to display. What they essentially say is, "Within the current subset of documents (in this case, a category), find the twenty-first document, and then display it and the nine documents that follow." Of course, more than ten could be displayed by increasing the Count parameter, but given the multi-line nature of the catalog entries, ten is a manageable number. I mentioned that there was a second use for these arguments; not surprisingly, they too are used by the WebQueryOpen agent to perform the navigational calculations.

Just to reiterate: the WebQeryOpen agent in this example has no effect in determining which and how many documents Domino displays; that is accomplished by the parameters on the URL. The agent *does* have an effect on the overall navigation, however, because it is responsible for building the links that will command Domino to traverse a given set of documents. The URL above is the product of the WebQueryOpen agent. If the agent were eliminated, Previous and Next links would have to be included as part of the Catalog View Single Category form.

The (ViewCat WebQueryOpen) agent is well-documented, and therefore I'll simply give a brief overview of its functionality and leave it to you to explore in depth. You can go to the **(ViewCatWebQueryOpen) agent** sidebar to see the complete agent.

The agent begins by parsing apart the URL parameters. With the results, the agent can go after the most crucial piece of information: the total number of documents in the chosen view/category. Once this total is determined, the agent can then use the Start and Count parameters to figure out where within the overall total the customer is browsing. That information is then written to the page, along with the links for subsequent navigation, where appropriate.

To see how this happens, take another look back at the Catalog View Single Category form (ViewCat) that was included as part of the previous example. Notice the Prev, TotalCount, and Next fields immediately above the embedded view and also their duplicates below the view, which have been included as a matter of convenience to the customer. These are the computed-for-display fields that the WebQueryOpen agent writes to once it has determined the links and information to display. If you refer back to the screenshots earlier in this example, you'll see that Prev displays the Previous 10 Items link, TotalCount displays the "Items x through x of x" information, and Next displays the Next 10 Items links. When there is no link to include (for instance, if a customer is at the beginning of a category, there is no need for a Previous link), one is not written to the document and its field will simply not be visible to the customer.

That's all there is to it! By adding a few extra fields to a form and including this

© Copyright 2000 Iris Associates, Inc.

agent in a database, customers can have a better sense of what they're grappling with as they delve into product categories. Did I mention that this technique is—wait, take a guess. That's right!—applicable across an array of Domino applications, not just e-commerce solutions.

Where next?

Clearly, navigation is a cornerstone of any e-commerce site, but it's only part of the story. Getting customers where they want to go is a noble endeavor, but enabling them to order products is a *lucrative* endeavor (notice how I didn't say profitable— that part is up to you.) In the next article, I'll explain how to get Domino to perform session tracking without using Domino's authentication mechanism, and I'll also dissect the item form, which allows customers to actually place catalog items in a shopping cart. In the third article, I'll delve into the shopping cart itself.

ABOUT THE AUTHOR

Michael Patrick is a Senior Consultant with **Knowledge Resource Group** in Indianapolis, Indiana.

© Copyright 2000 Iris Associates, Inc.

]

The (ViewCatWebQueryOpen) agent

Here is the code for the (ViewCatWebQueryOpen) agent:

```
Sub Initialize
	Dim s As New NotesSession
	Dim doc As NotesDocument
	Set doc = s.DocumentContext

	Call DoPrevNextHTML (doc)

End Sub

	Sub DoPrevNextHTML (doc As NotesDocument)

	' Figure out how many docs are about to be displayed in the view or view catagegory being shown, and
	' display where this set is in the view (e.g., docs 26-50 of 123), and display previous and/or next links as
	appropriate

	Dim db As NotesDatabase
	Dim v As NotesView, entry As NotesViewEntry

	Dim intTotal As Integer, intStart As Integer, intCount As Integer, intNextStart As Integer, intPrevStart As
	Integer
	Dim strCatParm As String, strNextlabel As String, intNextEnd As Integer, strNextParms As String,
	strPrevParms As String, strPrevLabel As String, intPrevEnd As Integer
	Dim intNext As Integer, intPrev As Integer

	Set db = doc.ParentDatabase
	Dim vPath As Variant

	'Set URL path for return
	vPath=Evaluate({@ReplaceSubstring (@Subset (@DbName; -1); "\\" : " "; "/" : "+")})

	'Parse out the four parameters passed in as part of the URL - View, Category, Start, and Count

	vView = Evaluate ({@Middle (Query_String + "&"; "&View="; "&")}, doc)
	vCat =		Evaluate ({@Replacesubstring (@Middle (Query_String + "&"; "&Cat="; "&"); "+" ; " ")}, doc)
	vStart = Evaluate ({@Middle (Query_String + "&"; "&Start="; "&")}, doc)
	vCount = Evaluate ({@Middle (Query_String + "&"; "&Count="; "&")}, doc)

	If vStart(0) = "" Then
		intStart = 1
	Else
		intStart = Cint (vStart(0))
	End If

	'Default Count to 25 if one wasn't provided on URL

	If vCount(0) = "" Then
		intCount = 25
```

© Copyright 2000 Iris Associates, Inc.

```
Else
	intCount = Cint (vCount(0))
End If

'Use strCatParm for the navigation links built later in the agent.  After the following code, it sould contain, for
example,
'"&Cat=Political+Thought"

If vCat(0) = "" Then
	strCatParm = ""
Else
	doc.tempcat = vCat
	vCat2 = Evaluate ({ @Replacesubstring (tempcat; " " ; "+") }, doc)
	strCatParm = "&Cat=" + vCat2(0)
End If

'Get the view passed on the URL

Set v = db.GetView (vView(0))

'If there is no category the user is browsing, simply count all the documents in the view.  If they are browing a
category,
'use the GetEntryByKey method of NotesView to get the first document in a category and then count its
siblings to get
'the total for that category

If vCat(0) = "" Then
	'doc.PrevNext = "equals blank"
	intTotal = v.AllEntries.Count

Else
	'doc.prevNext = "else"
	Set entry = v.GetEntryByKey(vCat(0))
	intTotal = entry.SiblingCount

End If

'Determine if Previous and Next links are neded, and what the proper Start parameter is for both

If intStart = 1 Then
	' No Prev button.  Next = start plus count
	intPrev = False
	If intStart + intCount > intTotal Then
		' No need for next button
		intNext = False
	Else
		intNext = True
		intNextStart = intStart + intCount
	End If

Else
	intPrev = True
	If intStart + intCount > intTotal Then
		' No next button
		intNext = False
	Else
		intNext = True
		intNextStart = intStart + intCount
	End If
	intPrevStart = intStart - intCount
	If intPrevStart < 1 Then intPrevStart = 1
End If
```

© Copyright 2000 Iris Associates, Inc.

```
    'Build the HTML (opening and closing tags) for both "Prev" and "Next" links - to be used once verbiage for
    each
    'link is decided upon

    strHrefStart = {<A HREF="/} + vPath(0) + {/ViewCat?ReadForm&View=} + vView(0) + strCatParm
    strHrefEnd = {</A>}

    'Build the verbiage for the "Next" link.  If there are less than Count items left to display, use "Last" instead of
    "Next"

    If intTotal - intNextStart < intCount Then
        strNextLabel = "Last " + Cstr (intTotal - intNextStart + 1) + " Items"
    Else
        strNextLabel = "Next " + Cstr (intCount) + " Items"
    End If

    'Build parameters for "Next" link

    strNextParms =  {&Start=} + Cstr (intNextStart) + "&Count=" + Cstr(intCount)

    'Build verbiage for "Previous" link.  If there are less then Count items left to display, use "First" instead of
    "Previous"

    If intStart - intCount < 0 Then
        strPrevLabel = "First " + Cstr (intCount) + " Items"
    Else
        strPrevLabel = "Previous " + Cstr (intCount) + " Items"
    End If

    'Build parameters for "Previous" link

    strPrevParms = {&Start=} + Cstr (intPrevStart) + {&Count=} + Cstr (intCount)

    'Write links to document if they are needed.  If not, these fields will not contain values and will not be visible to
    users

    If intPrev Then
        doc.Prev = strHrefStart + strPrevParms + {">} + strPrevLabel + strHrefEnd
    End If

    If intNext Then
        doc.Next = strHrefStart +  strNextParms + {">} + strNextlabel + strHrefEnd
    End If

    'Build 'Items x through x of x' information

    If (intStart + intCount - 1) > intTotal Then
        intShowing = intTotal
    Else
        intShowing = intStart + intcount - 1
    End If

    'Write it to document

            doc.TotalCount = "Items " + Cstr (intStart) + " through " + Cstr (intShowing) + " of " + Cstr (intTotal)

    'Duplicate the links (provided there are any) and information below the view as well

    doc.Prev_1 = doc.Prev
    doc.Next_1 = doc.Next
    doc.TotalCount_1 = doc.TotalCount
End Sub
```

© Copyright 2000 Iris Associates, Inc.