Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today

Level: Intermediate
Works with: Designer 4.6
Updated: 02-Feb-98

by
Mark Gordon
and
Andrew Broyles

*[Editor's note: This is the first article in a two-part series on how to add "friendlier" user interface features to your Web applications. You'll learn how forms and views work together on the Web and how to use them to add a quick-find search bar to a Web application. The **next article** will focus on how to create views that allow users to perform actions on multiple documents, and even change keyword fields on multiple documents.]*

Getting your application out on the Web can be a snap, because Domino does so many things automatically for you. Adding the features that make your application "snappy" to use, however, is a different story. This article will focus on features you can add into your views -- where your users spend much of their time navigating -- to make them more friendly. Many of the things we'd like to add are features that the Notes client supports, but which don't show up automatically on the Web, such as the quick search dialog that pops up as you type into a view, or the ability to select multiple documents from a view. We'll bring these things "back" to Domino Web applications, as well as add other features, by integrating forms with our views to produce powerful new UI constructs.

This is the first article in a two-part series that will demonstrate this technique of combining forms and views in your Web applications. In this series, you'll learn how to:

- Add a quick-find search bar to the top of your Web views, so users can "jump" down to the part of the view that they want.
- Create a Web view that allows users to select multiple documents, and perform some action on them.
- Create a Web view that allows users to edit status codes or other keyword values on documents *directly from the view.*

This time, we'll focus on showing you how to implement the first feature -- the quick-find search bar -- while introducing the concepts and techniques involved in building all three features. In the **next article**, you'll learn how to create the second two features. All the examples we'll discuss are working examples you can try by **downloading the sample database** in the Sandbox.

The article is written with the experienced Notes/Domino developer in mind. We assume you already know how to design forms and views for Domino Web applications, and that you have some understanding of HTML. These techniques require no knowledge of Java or JavaScript.

Note: If you're using the 4.6 database property "Web access: Use JavaScript when generating pages," you'll need to modify our technique for adding the Find button to the top of a Web view. The reason is that

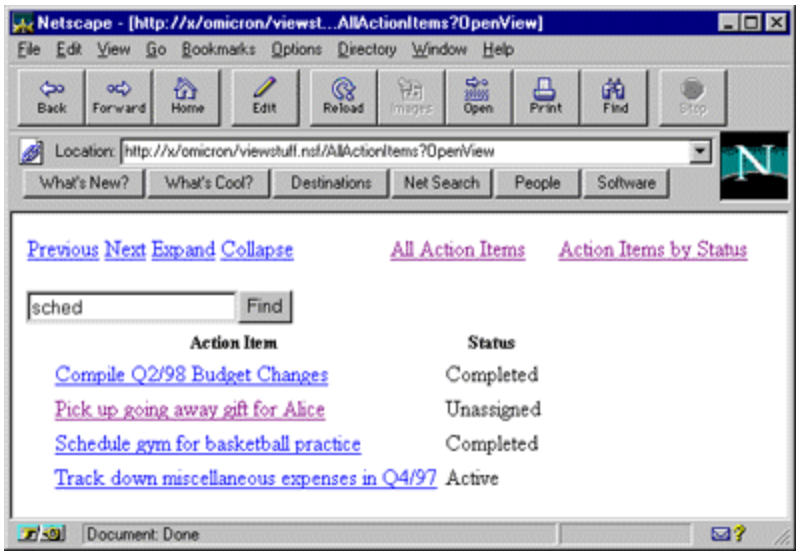
when you have the property enabled, Domino generates its <Form> tags differently -- that is, it generates a <Form> tag even for $$ViewTemplateDefault forms. The solution is to put a </Form> tag before your first <Form> in your HTML. Then, the Find button and other form-within-view constructs will work.

The Quick Find tool

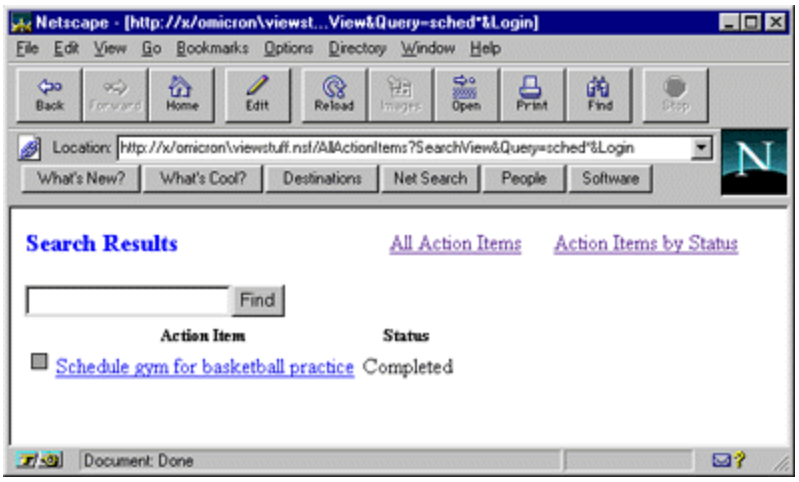
One of the "friendly" UI features that you might take for granted in your Notes application is that a user can just begin typing in any view, and a Quick Search dialog automatically appears. But, when you move this application to the Web, you lose this helpful functionality. The reason, as you may already know, is that Notes views and Domino views are very different animals. There is no HTML construct for a view; a Domino view shown on the Web is simply an HTML table of links. And, of course, Domino generates those dynamic links for us.

So, to get the quick-search functionality into your Web application, you can't rely on the standard Domino view. However, what you can do is combine a form and a view on the same Web page. HTML has a native form construct, the syntax for which Domino generates for you based on the design of the form in the Notes database. And, in HTML, any page may contain a form, or more than one form.

Our solution then is to design a Web page that puts a form at the top of a view. The form contains a field and a Find button for users to do searches. Here is what the Quick Find looks like at the top of a view:



The user can simply type in a few letters and click Find (or press Enter). Domino then displays a list of action items that matches the criteria. The above example will look for all action items that start with "sched."

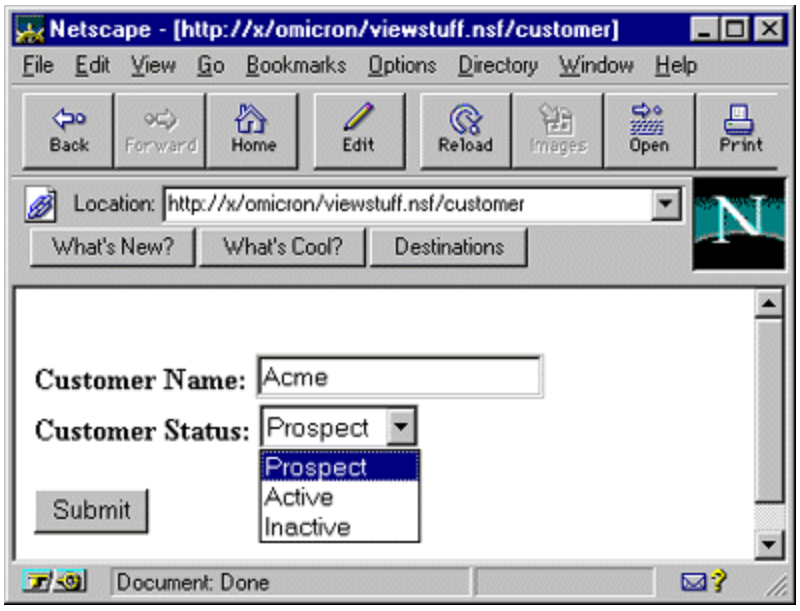


The search results are presented in a format similar to that of the view. The user can click on any of the resulting documents to open one, type in another search string, or go to another view.

How the Find button works

As stated earlier, the Find button appears in a form at the top of the All Action Items view. To understand how the Find button works, we'll look at how forms are coded in HTML and how Domino generates them. As you'll see, you don't need to do much HTML coding, because you can "steal" the HTML from Domino-generated forms. But you do need to understand how the HTML works.

Here is a very simple Notes form shown from the Web, and below it, the HTML that Domino generated to display the form to a Web browser.




```
<HTML>
<!-- Lotus Domino Web Server (147.26a  - Sep 15 1997 on Windows NT/Intel) -->
<HEAD>
</HEAD>
<BODY TEXT="000000" BGCOLOR="ffffff">
```

```
<FORM METHOD=post ACTION="/omicron/viewstuff.nsf/4aa58b19197cd6b10525657
d00412959?CreateDocument" ENCTYPE="multipart/form-data" NAME="_Customer"><BR>
<B>Customer Name: </B>
<INPUT NAME="Name"><BR>
<B>Customer Status: </B>
<SELECT NAME="Status">
<OPTION>Prospect
<OPTION>Active
<OPTION>Inactive</SELECT>

<P>
<INPUT TYPE=submit VALUE="Submit"></FORM>
```

Notice the HTML that specifies the form is everything between the <FORM> and </FORM> tags. (There could be any number of other things defined on this HTML page before or after these FORM tags.) This form is defined with two fields on it. The first is an editable field, called "Name" -- which in HTML, is an INPUT field. The second field is a combo box called "Status" -- a combo box is a SELECT field in HTML.

When the user clicks the Submit button -- defined by the HTML code <INPUT TYPE=submit VALUE="Submit"> -- a data stream is sent back to the Web server via a form *Post Action.* This data stream contains the field names and their values, led by a URL that tells the server to create a document. This command was specified in the ACTION we saw above:

```
<FORM METHOD=post ACTION="/omicron/viewstuff.nsf/4aa58b19197cd6b10525657
d00412959?CreateDocument"
```

Domino is generating HTML to define the form, and is also generating HTML for the browser to tell Domino what to do with the form once it's submitted. Those of you familiar with Notes applications, but new to Web development, may find this odd. In effect, Domino is saying to the browser, "Here is the form. When you submit it back to me, I won't remember what to do with it, so I've included a note to myself to tell me what to do with it. Please return the note along with your submitted data."

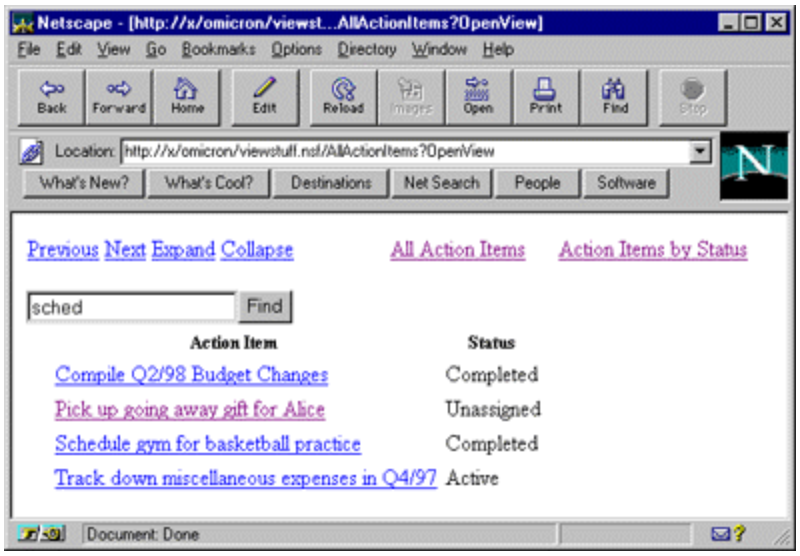
At this point, you may wonder if Domino knows what to do with the data, since it opened the session with the browser in the first place. The answer is no, it doesn't. Unlike a Notes client/Domino server session, there is no "session" between a Web client and a Web server. There are only transactions. Every time a browser sends a URL to a Web server -- whether that URL is requesting a page or submitting a form -- the browser also sends login information it has cached (if login is required) to identify the user. Since there is no session, the browser has to send the server not only the data, but also the instructions on what that data is and what should be done with it.

The above "FORM METHOD=post ACTION" action tells Domino to create a document in the viewstuff.nsf database using the Customer form. (Domino generated the POST action with the Note ID of the Customer form rather than the name "customer." As you'll see, when we code our own form and POST action, we can use the name instead.)

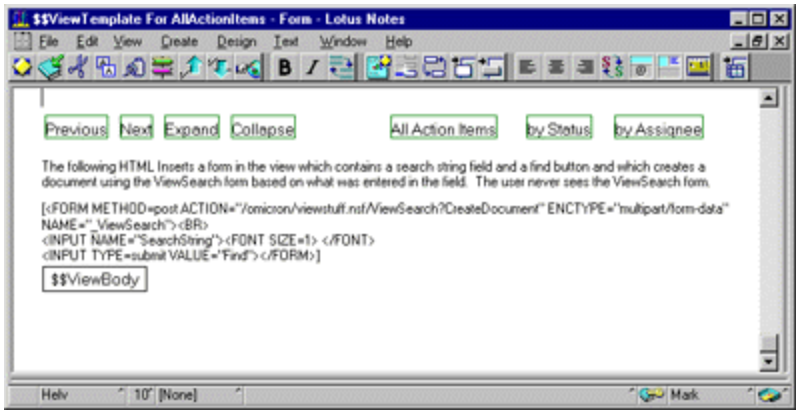
The point in describing all this is that there are two "forms" that come into play here, one for the Web client and one for the Web server. The first is the HTML form, into which the user enters data using a browser; and the second is the form in the Notes database that Domino invokes to create the document. They are usually the same, in that Domino uses the Notes form to generate the HTML form and then uses the same Notes form to collect the data sent back with the Submit button (the form POST action).

But the HTML form does *not* have to be generated by Domino. You can code an HTML form and put it in anywhere you like, including on a view (using a $$ViewTemplate form). You just need to make sure there is a Notes form with the same name and with fields named appropriately to collect the data. (You'll get an error if you don't have all the same fields on the Notes form.)

Now, back to the Quick Find example. Here again is the view as the users see it:

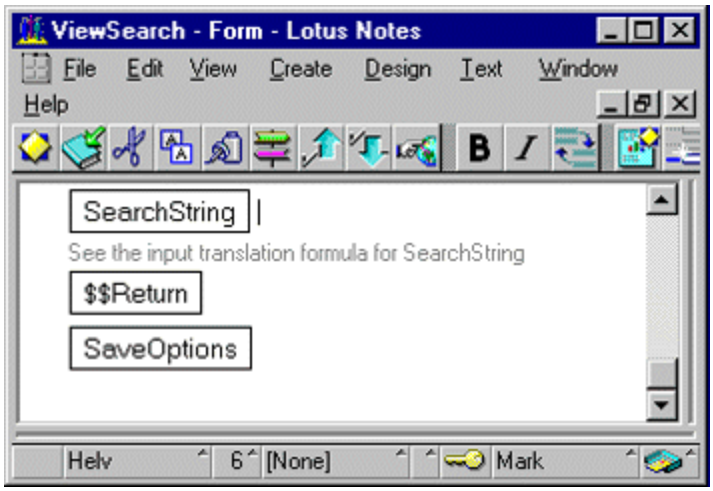


To design this view with the Find button at the top, we use a $$ViewTemplate form. Here is the "$$Viewtemplate for AllActionItems" form:



Notice that the $$ViewTemplate form includes the HTML for the search form. However, we'll show you later how you can put the HTML in a computed-for-display field, so you can use variables rather than hard-coding everything on the form.

The HTML defines an unlabeled field called "SearchString," and right next to it the Submit button, labeled "Find." Notice we have instructed Domino to use the viewstuff.nsf database and theViewSearch form to create a new document with the data that we send in the SearchString field. Of course, we are not really creating a document we want to save in the database. Rather, we are creating a temporary ViewSearch document for the purpose of computing a search query to search the database; we won't save the document. Here is the ViewSearch form, *which the user never sees*, but which Domino uses when accepting our submitted search:



Notice that the form includes a field to capture the search string and a $$Return field to execute the search query via a URL full-text search. The editable SearchString field accepts the value submitted in the HTML form field with the same name. However, we use the following input translation formula to modify the value:

```
@ReplaceSubstring (SearchString + "*"; " "; "+")
```

This formula adds an asterisk to the end of whatever the user typed, and then replaces any spaces with plus signs. Adding the asterisk lets the user type in a partial word and get a match. So, when the user types in "sched," the full-text search engine actually searches for "sched*" and returns any matching action items. We replace any spaces in the query with plus signs, because as you'll see in the $$Return field formula, we are passing a full-text search to Domino on the URL, and we cannot use spaces in URLs.

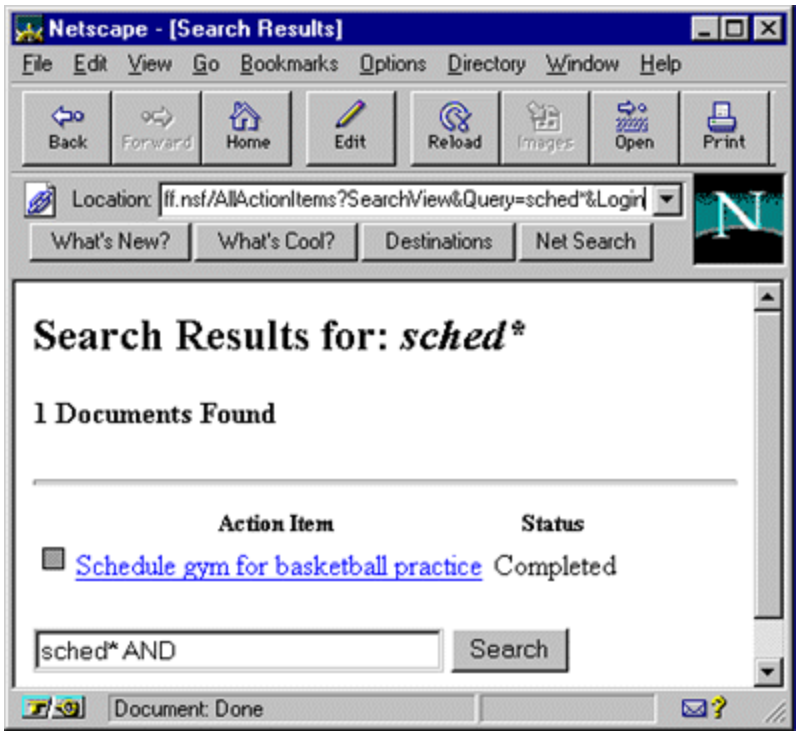
Here is the formula for the $$Return field:

```
database := @Subset (@DbName; -1);
"[/" + database + "/" + "AllActionItems"
+ "?SearchView&Query=" + SearchString + "&Login]"
```

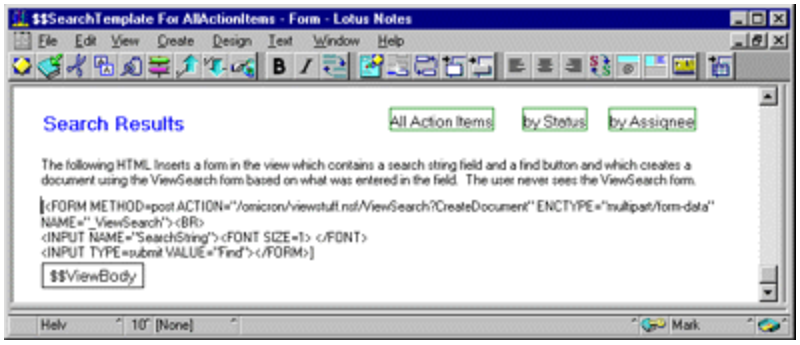
This formula builds a URL that tells Domino to search the AllActionItems view for the SearchString. The &Login at the end is generally not needed, but is sometimes necessary if the database being searched has restricted access. It does *not* cause the user to be prompted to log in again.

The search results

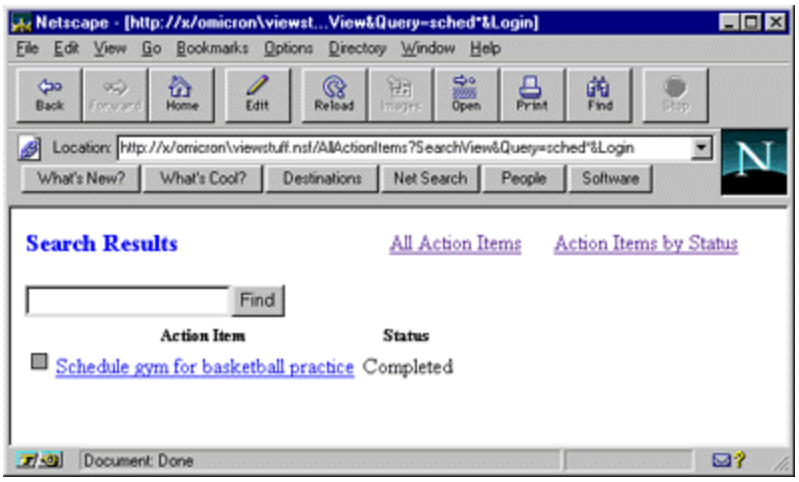
Now, let's look at what our Quick Find returns. Again, we'd like for Web users to have the same functionality that Notes users have. This means, we'd like them to feel like they're still "in the view" after the search runs. By default, Domino returns search results in the following format:



Again, we'd prefer to customize the results so that users feel like they're still in the view, and can either do another search, return to the full view, or go to another view. So, we'll design a "$$SearchTemplate for AllActionItems" form that overrides the default search results. It contains the search HTML again, as well as links to other views:



With this form in place, the user now sees search results that look like this:



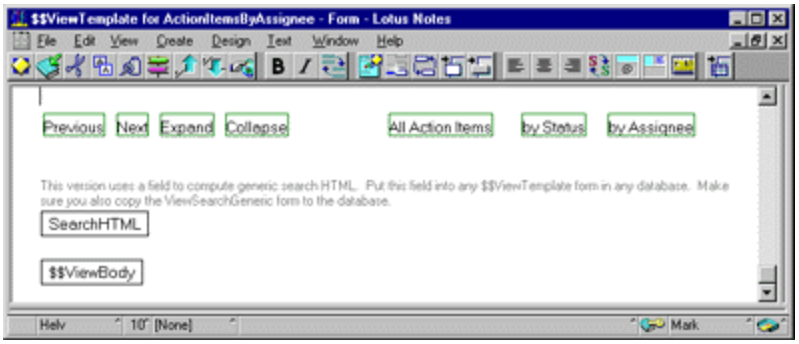
Less HTML coding, more reusability

There are a couple of things we can do to make this technique a bit easier. The first is to avoid coding the HTML from scratch, like we did earlier. Even though we need to use HTML on our $$ViewTemplate to define the ViewSearch form (the search field and Find button), we can "cheat" and have Domino generate the HTML for us. We simply design the ViewSearch form in Notes *first*, before we create the $$ViewTemplate form. On the form we put the editable SearchStringtext field and a hotspot button labeled "Find" to the right of the field. We can then open that form from a browser (by typing xx/viewstuff.nsf/ViewSearch at the URL) and "steal" the HTML that Domino generates.

As you probably know, most Web browsers allow you to view the underlying HTML on any Web page, generated by Domino or otherwise. Simply choose View - Document Source from the Netscape Navigator menu bar, or View - Source from the Internet Explorer menu bar. You can copy any portion of the HTML and use it on your own form. You can then create the $$ViewTemplate form, and paste this HTML onto it above the $$ViewBody field.

The step that will really save you time is to make the view search HTML generic so that it can be used on any $$ViewTemplate form in any database. To do this, we put the HTML into a computed-for-display field rather than directly on the form. Of course, when you use HTML in a field formula, you need to put the static portions of it in double quotes. If the HTML you are using already has double quotes for its syntax, you need to put a backslash in front of any HTML double-quote so the Domino formula interpreter understands that it should ignore them. This makes for harder-to-read, but much more reusable HTML.

The "$$ViewTemplate for ActionItemsByAssignee" form contains a computed-for-display field called "SearchHTML" that can be used as is for any view in any database:



Here is the formula for the SearchHTML field:

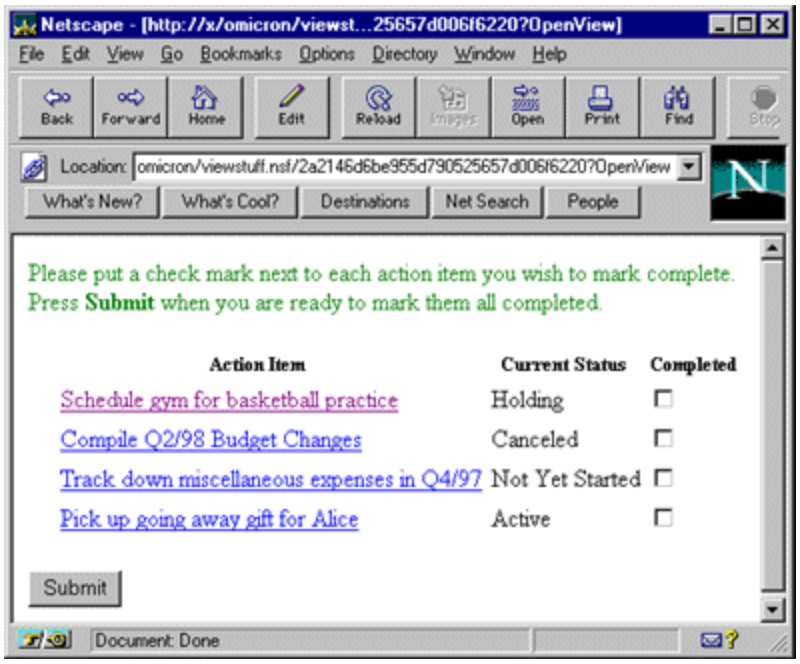
REM "Inserts a form in the view which contains a search string field and a find button and which";

```
REM "creates a temporary document using the ViewSearchGeneric form based on what was entered in the field.";
REM "The user never sees the ViewSearchGeneric form.";

REM "This HTML is generic; it calculates both the current database ... ";
db := @Subset (@DbName;  -1);

REM "... and the current view being searched from.";
REM "We calculate the current view here and store in variable VT for use in the HTML below.";
REM "Given what @ViewTitle returns, we replace spaces with plus signs and hyphens with backslashes.";
REM "We do this so that the view name can be used in the URL which will be computed in the $$Return field";
REM "on the ViewSearchGeneric form. Spaces don't work in URLs, and since @ViewTitle returns a hyphen";
REM "instead of a backslash for cascaded views, we put the hyphen back in.";
VT := @ReplaceSubstring (@ViewTitle; " ": "-" ; "+":"\\");

REM "The view name we stored in variable VT is passed to the ViewSearchGeneric field form in a hidden-by-HTML";
REM "field called ViewToSearch, which is defined below in the HTML.";
REM "The ViewSearchGeneric form also has a field called ViewToSearch, which is used by the $$Return";
"<FORM METHOD=post ACTION=\"/" +
db +
"/ViewSearchGeneric?CreateDocument\" ENCTYPE=\"multipart/form-data\">" +
"<INPUT NAME=\"SearchString\">" +
"<INPUT NAME=\"ViewToSearch\" VALUE=\"" + VT + "\" type=hidden>" +
"<INPUT TYPE=submit VALUE=\"Find\">"
```

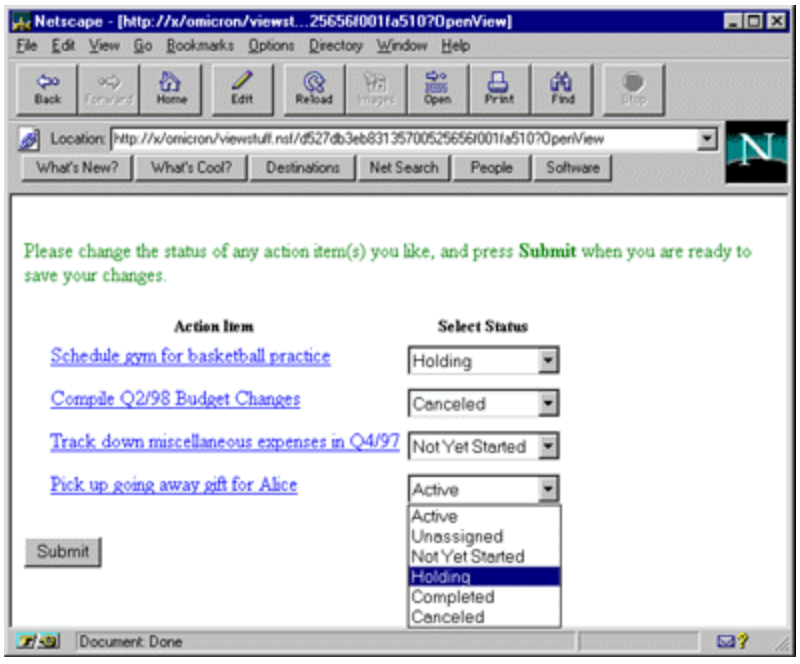
The tricky part of this formula is calculating the name of the view so that we can execute a full-text search against the proper view. The view name is available to us while the view is open (on the $$ViewTemplate form using @ViewTitle). However, what the user types in is passed to Domino on the ViewSearchGeneric form, where the search URL is built, and the @ViewTitle formula doesn't return anything on that form. So we've added a ViewToSearch field on the HTML form -- hidden via HTML because it won't work with a Notes hide-when formula -- and used dynamically computed HTML to set the view title into the ViewToSearch field. We've also included this ViewToSearch field on the ViewSearchGeneric form, which accepts the value passed to the server from the browser.

Next article: Multiple selects and status changes from a view

Now we've seen how forms and views work together on a page, and described how to add a Quick-Find area to the top of your views. In the next article, we'll see the real payoff for learning how to combine forms and views, by using these techniques along with some others to create views that let you select multiple documents from a view, like this:



We'll also describe how to create a view that let's you actually *change a keyword field* on multiple documents all from a view, without having to open, change, and submit each document. This is something you can't even do with a Notes client! Here is an example:



Can you think of any applications you've worked on where you'd like your users to make multiple choices directly from a view, or to change status codes from a view in a workflow application? See you next time!

ABOUT ANDREW

Andrew Broyles is president and lead architect of Replica Inc., a Lotus Business Partner in Indianapolis, Indiana. Replica specializes in integrating disparate legacy and client-server data sources in Lotus Notes/Domino.