Adding a friendly ad-hoc query tool to Domino applications

by Mark Gordon

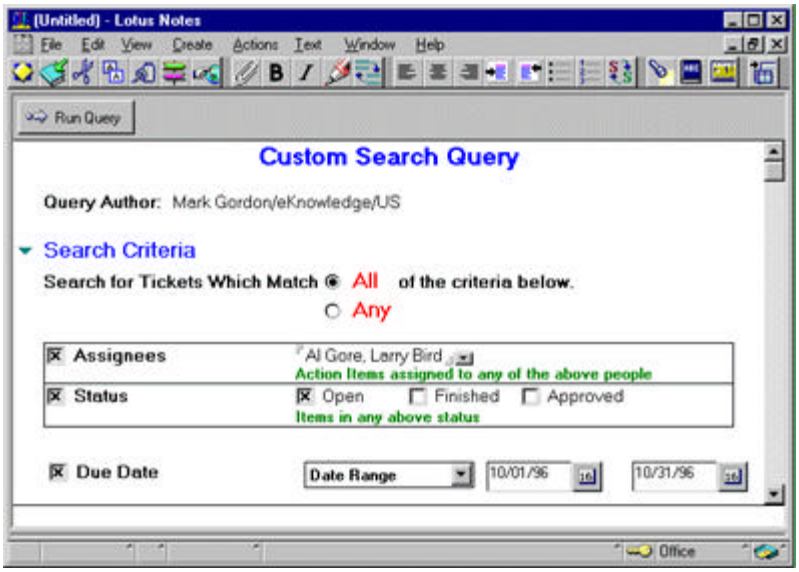
[Editor's note: This article resides in "Notes Today", the technical Webzine located on the http://www.notes.net Web site produced by Iris Associates, the developers of Domino/Notes.]

What do you tell your users when they ask you how they can run ad-hoc queries against the Notes application you've delivered to them? Most Notes developers struggle with this issue at some point -- the more valuable the information that users accumulate in a Notes database, the more likely it is that they'll want to selectively find that information. This article describes a technique for building a powerful, easy-to-use, ad-hoc query capability into any Domino 4.5 application. The approach uses techniques taught in the Lotus Notes and Domino 4.5 Application Development 1, Application Development 2, and LotusScript courses. Anybody who has taken these classes, or has similar knowledge, should be able to follow the techniques presented here.

Many people don't know this, but the Notes full-text search engine can handle much more than just keyword searches. Here is an example of a full-text query that will find in an Action Items database all the "Open" action items assigned to either "Al Gore" *OR* "Larry Bird" and due in "October 1996":

FIELD FORM = Action Item AND ((FIELD AssignedTo = Al Gore OR FIELD AssignedTo = Larry Bird) AND (FIELD Status = Open) AND (FIELD DueDate >= 10/01/96 AND FIELD DueDate <= 10/31/96))

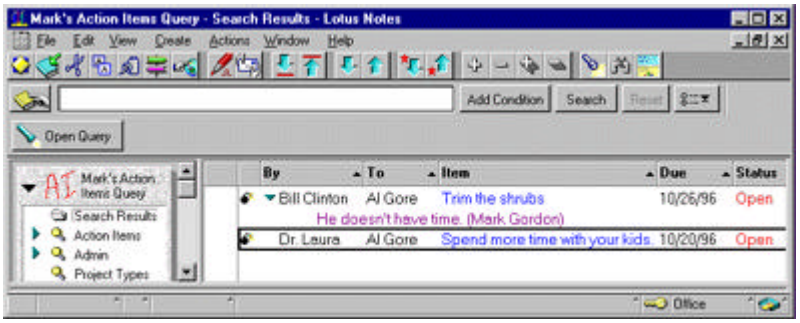
If you enter the above query into the full-text search bar, you will get a list of action items returned that match those criteria. Just teach your users the syntax, right? Although the search engine is powerful, the only built-in way to point and click your way to that kind of query is with the full-text Search Builder (which comes up when you click Add Condition on the full-text search bar). Users may have a difficult time finding query by form. So the first part of our solution involves designing a simple yet powerful form that lets the users point and click their way to a query, and then *generates* the syntax you see above. After generating the syntax, you can then use LotusScript to pass the query to the full-text search engine -- so the user never has to deal with the query syntax at all.



Then what? If I search an action item database with the criteria in the example above, I would like to see the action items that match the criteria and any response documents. The native full-text search engine displays documents sorted either by relevance (how many times the items you were searching for occurred in each document) or by date. However, response documents don't show up under their parents. If I typed the above query into the full-text search bar, I would get action items, but would have to clear the search results to see the response document for a

particular action item. Our solution is to do something to make the search results more useful -- by putting them into a folder.

Within LotusScript, when you execute a full-text search, you can take the resulting documents and put them into a personal folder. (A current LotusScript restriction is that you cannot move documents into a "Personal on first use" folder unless the folder has already been created.) So we'll create a Search Results folder for users that is shared and "Personal on first use," designed after an Action Items and Responses view. Our script takes the search results and puts them in the folder (each user has his/her own version of the folder to avoid overwriting each other's search results). If the Search Results folder is set up to show a response hierarchy, Notes automatically puts the responses to any action items matching our criteria into the same folder as the action item, as shown in the following screen. Once we have put the results into the folder, our final step is to open the folder on the user's desktop.



We'll use the following steps to create the ad-hoc query capability.

1. Decide which fields you want the users to choose from to create a query.
2. Design a Custom Query form that lets the user pick and choose which fields -- we'll call them filter fields -- and which values or ranges to specify.
3. Add hidden computed-for-display fields, one corresponding to each filter field, which calculate the full-text search syntax for each filter field.
4. Add a computed field that puts the syntax for all the filter fields together (separated with ANDs or ORs) into a complete query.
5. Add a Run Query action button, which does the following:
    - Clears the user's Search Results folder from the results of the prior search.
    - Calls a LotusScript agent that runs the query and puts the results in the Search Results folder.
    - Opens the Search Results folder.

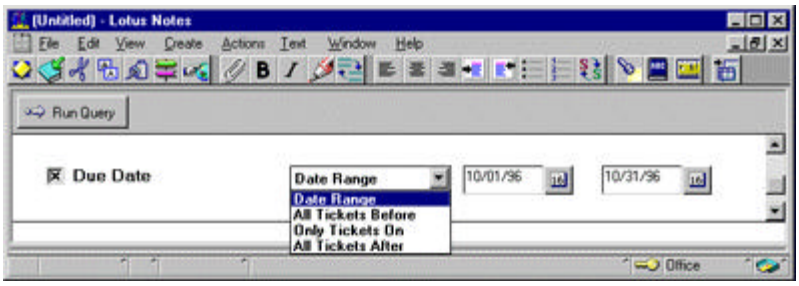
Step 1. Decide which fields will be filter fields

Our example uses an action items database. Each action item tracks the assigner, the assignee, the date assigned, the date due, the subject and description of the action item, and the status of the action item. We probably want the user to be able to search for action items based on any of the above criteria. For simplicity in our example, we'll use just the three fields we've shown in the previous figures: "Assignee," "Status" and "Due Date."

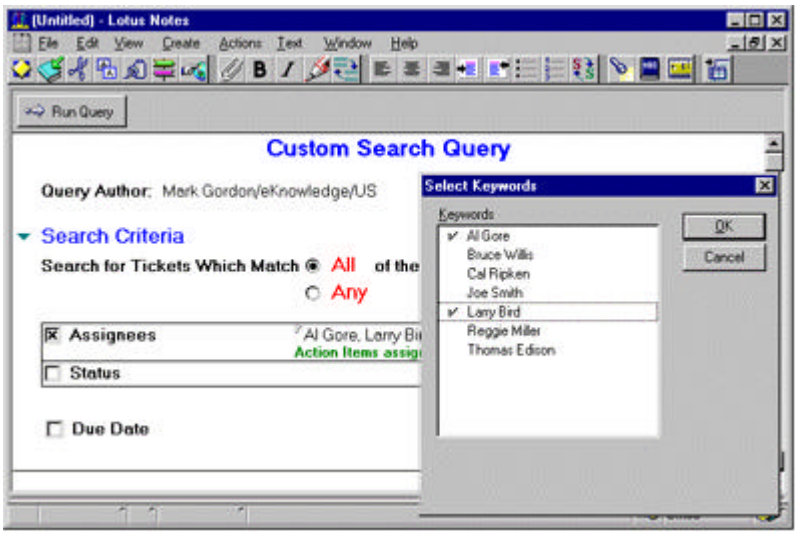
Step 2. Design the Custom Query form

Our users will create a query using a Custom Query form, specify the results on that form, and then run the query. The query itself is then saved, and the user can go to his/her own personal Custom Queries view (a "Shared/Personal on first use" view that shows each user his/her own queries) to look at or reuse any query created in the past. The query form is where the query is specified and also where the syntax is generated (although the user may never see that syntax).

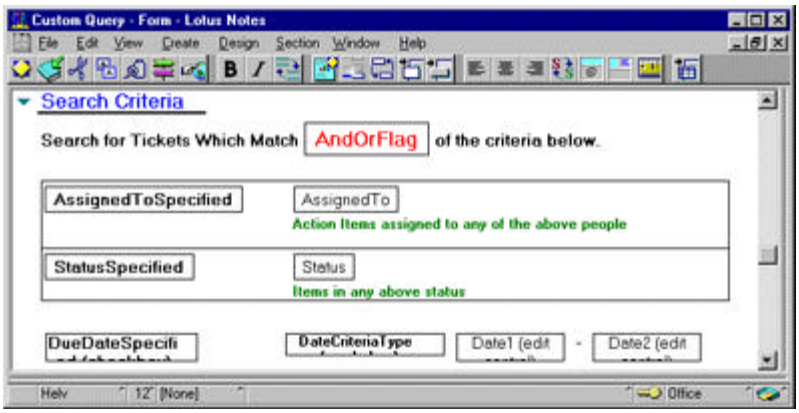
We could clone the Action Item form, which is already used to enter and read action items, and use it as the basis for our query form -- sort of a query-by-example. The problem is that not all of the fields on the Action Item form may make sense as potential search filter fields. And those that do may need to be displayed differently when used as filter fields. For example, on the Action Item form the "Due Date" field may be editable or calculated, but either way it's defined as a single date. In our Custom Query form, however, we want the user to be able to filter based on his/her choice of the following: all action items due *before* a certain date, all action items due *after* a certain date, or all action items due *within a range* of dates, as shown in the following screen.



So we design a new form, which shows the possible filter fields. The user does not need to filter using all the possible fields; and by using hide-when formulas, we show only selection boxes for the criteria the user picks. In the previous screen, the user is specifying a date range to filter on the Due Date. But in the following screen, the user is filtering only by Assignee, so the Status and Due Date selection areas are blank.



The form has a checkbox next to each filter field, to let the user decide whether to specify criteria for that filter field. The filter field itself contains whatever keywords formulas are required. For example, the Assignees checkbox corresponds to the AssignedToSpecified field. When the user selects the Assignees checkbox to filter on assignees, the AssignedTo field displays (driven by a hide-when formula). In the AssignedTo field, the user can then choose from a list of assignees that is determined by an @DBcolumn formula. The previous figure shows what this process looks like to the user, and the following figure shows the same portion of the form in design mode.



Step 3. Generate the full-text search syntax for each filter field

Now we need to generate the full-text search syntax based on these criteria. To do this, we can use the @Explode and @Implode functions, which make the AND and OR syntax easy to generate.

The syntax for finding assignees Al Gore and Larry Bird is the following:

FIELD AssignedTo = Al Gore OR FIELD AssignedTo = Larry Bird

So I created a field called "AssignedToSyntax," computed for display, with the formula:

```
rem "Insert FIELD ASSIGNEDTO = in front of each assignee in the list";
vlist := "FIELD AssignedTo = " + AssignedTo;
rem "
@If (AssignedToSpecified = "Yes" ;
@Implode (vlist; " OR ");
NULL)
```

The variable *vlist* becomes a list, with two values: "FIELD AssignedTo = Al Gore" and "FIELD AssignedTo = Larry Bird." The @Implode function turns *vlist* into a string separated by the literal string "OR." If the AssignedToSpecified checkbox is not selected (the user didn't specify any assignees to search by), the whole formula returns NULL -- there is no assignees criteria.

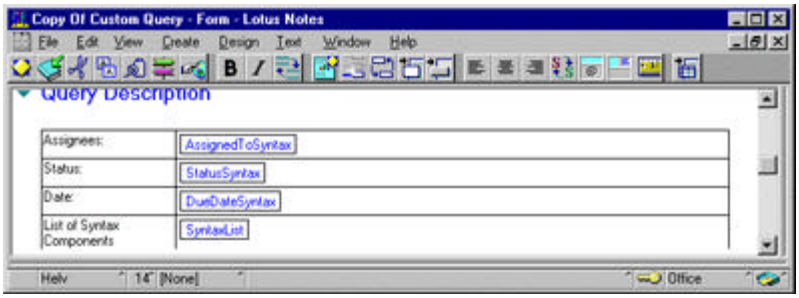
The "DueDateSyntax" formula is a little trickier, because we have an additional field, DateCriteriaType, in which the user specifies a date filter. Also, we use two Date fields if the user is specifying a range of dates. Here is the formula for DueDateSyntax:

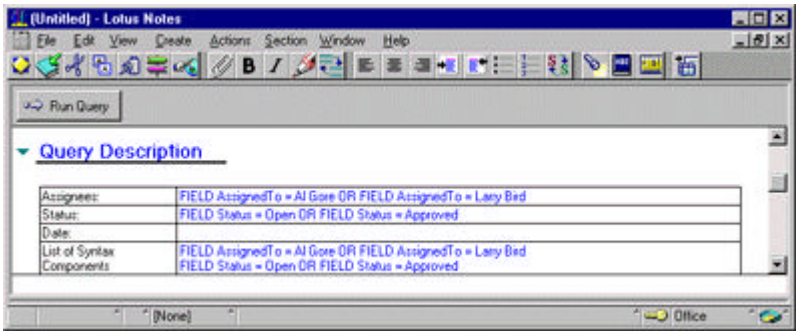
```
rem "Convert the dates to text";
Date1Text :=  @Text (Date1);
Date2Text := @Text (Date2);
rem "Generate the appropriate syntax, depending on whether the user chose a RANGE, or BEFORE,";
rem "ON, or AFTER settings";
@If (
  DueDateSpecified != "Yes" ;
     @Return (NULL);
  DateCriteriaType = "Range";
     "FIELD DueDate >= " + Date1Text + " AND " + "FIELD DueDate <= " + Date2Text;
  DateCriteriaType = "Before";
     "FIELD DueDate < " + Date1Text;
  DateCriteriaType = "On";
     "FIELD DueDate = " + Date1Text;
```

```
  DateCriteriaType = "After";
    "FIELD DueDate > " + Date1Text;
  "Error in date criteria"
)
```

If you do a syntax-calculated field like the examples above for every filter field, you end up with a set of search syntax snippets that you either AND together or you OR together (depending on whether the user picked ANY or ALL with the "Search for tickets that match ANY/ALL" radio button) to get the final search syntax.

We have three filter fields: AssignedTo, Status, and DueDate. So I used three computed-for-display syntax fields: AssignedToSyntax, StatusSyntax, and DueDateSyntax, as described above. The following figures show how these fields are laid out in Design mode (the layout matters only for your ease of testing, because once you have the formulas working you will want to hide them so they don't show up for the users), and then how the fields appear when in run mode.





Step 4. Combine the results into a complete full-text query

The SyntaxList field is simply a concatenation of the three syntax fields into a list. And the final field is QuerySyntax, which calculates the full syntax, including the restriction to search only for action items. It also includes parentheses around each criteria and ANDs or ORs between each criteria. The field is computed, rather than computed-for-display, because it will be stored with the Custom Query document and picked up by the LotusScript agent that runs the query. Here is the formula for the QuerySyntax:

```
rem "Put parentheses around each item in the SyntaxList";
ListWithParen := "( " + SyntaxList + " )";
rem "Put a space on either side of the AND or OR the user chose as the separator";
ExplodeParm := " " + AndOrFlag + " ";
rem "Start with the restriction to search only action items, and AND it with the other criteria";
"FIELD FORM = Action Item AND (" +
@Implode (ListWithParen; ExplodeParm)
+ ")"
```

The query that produced the syntax components shown in the previous figure generates the QuerySyntax field as follows:

FIELD FORM = Action Item AND ((FIELD AssignedTo = Al Gore OR FIELD AssignedTo = Larry Bird) AND (FIELD Status = Open OR FIELD Status = Approved))

Step 5. Add a Run Query button to the Custom Query form

After choosing the criteria, the user can click Run Query to run the query and show the results. The button must do the following:

- Clear the user's Search Results folder from the results of the prior search.
- Call a LotusScript agent that runs the query and puts the results in the Search Results folder.
- Open the Search Results folder on the user's desktop.

Note: Another way to store per-user data that actually has some performance benefits would be to use profile documents. These are special data objects, stored in the database, but they exist as design elements so that they never appear in any view. They are accessed by either one or two "keys," and they are cached in memory on servers. This means that you can store per-user data on a profile document, indexed by the user's name (and by an application name as well if you need per-user, per-application persistent data). When you access a profile document from LotusScript (using the NotesDatabase.GetProfileDocument method), it is returned to you as a regular NotesDocument instance, so you have the full range of scripting capability as for ordinary data documents. However, the lookup time is significantly faster than reading and writing environment variables.

Here is the formula behind the button:

```
rem "Save the Custom Query, causing the syntax calculation fields to recalculate";
@Command ([FileSave]);
rem "Set the unique doc ID of the Custom Query document into the notes.ini so the script agent can pick it up.";
@SetEnvironment ("AICustomQuery"; @Text (@DocumentUniqueID));
rem "Let the user know that the search results from the last query run are being cleared.";
@Prompt ([OK]; "Clearing Search Folder"; "The Search Results folder is being cleared, and the results of this new search will be placed there.");
rem "Open the Search Results folder and clear it";
@Command([OpenView]; "Search Results");
@Command([EditSelectAll]);
@Command([RemoveFromFolder]);
@Command([FileCloseWindow]) ;
rem "Run the query and then open the Search Results folder again";
@PostedCommand([ToolsRunMacro] ; "(Execute Custom Query)");
@PostedCommand([OpenView]; "Search Results")
```

The @commands are the most efficient way to clear the prior search results, but LotusScript is required to execute the query and put the new results in the folder. And @commands once again are the best way to navigate the user to those search results. So this button uses a combination of formula language and a LotusScript agent. Here is the code for the agent:

```
Sub Initialize

    ' Run a Custom Query using the syntax stored in a Custom Query document, and put the results
    ' in the Search Results folder

     ' Declare the session, DB, and document objects
```

```
    Dim Session As New NotesSession
    Dim DB As NotesDatabase
    Dim coll As NotesDocumentCollection
    Dim SearchDoc As NotesDocument

    ' Get the current database
    Set DB = Session.CurrentDatabase

   ' The search string is the full text search string generated for the user in the Custom Query document
   ' The doc ID of the last Custom Query run was stored in the notes.ini by the Run Query action button
    Dim SearchDocID As String
    SearchDocID = session.GetEnvironmentString( "AICustomQuery" )
    Set SearchDoc = db.GetDocumentByUNID ( SearchDocID)
    Dim SearchString As String

    ' The query syntax was calculated on the form
    SearchString =  SearchDoc.QuerySyntax(0)

    ' Check to see that db is FT indexed
    ' --if it is not, then exit the subroutine
    If Not db.IsFTIndexed Then
        Messagebox "Sorry.  The Action Items database is not full-text indexed.  Please contact the Notes
administrator."
        Exit Sub
    End If

    ' Run the search against the db and put the results in the folder
    Set coll = db.FTSearch (SearchString, 0)
    Call coll.PutAllInFolder( "Search Results" )

End Sub
```

Other enhancements

There are a number of other things you can do to make this ad-hoc query solution more powerful. One enhancement you can make is to generate an English-language equivalent to the query. For example, this syntax:

FIELD FORM = Action Item AND ((FIELD AssignedTo = Al Gore OR FIELD AssignedTo = Larry Bird) AND (FIELD Status = Finished))

can also be generated into a separate field (the title of the query), as follows:

Assignees (Al Gore or Larry Bird) AND Status Codes (Finished).

Now the user can find queries in the Custom Queries view with titles that are easier to understand. The same techniques apply to generating this syntax that we used to generate the full-text syntax.

Another enhancement you can make is to give users a choice of search results formats. For example, they might like to see action items matching their criteria, but categorized by Priority or by Assignee. You can prompt them for a results format, have a pre-defined folder for each format, and simply put the results into the appropriate folder.

Another idea is to allow searches with criteria that are on different forms within a database. Or you might tackle a multi-database search tool -- although the use of folders to show the results wouldn't work.

Even without enhancements, experienced Notes developers should find the ad-hoc query solution discussed in this article fairly simple to implement and extremely useful for users of almost any Notes application.

ABOUT THE AUTHOR

Mark Gordon is a certified Notes instructor and a Notes consultant. He teaches for Knowledge Development Centers, a Lotus Authorized Education Center in Indianapolis.