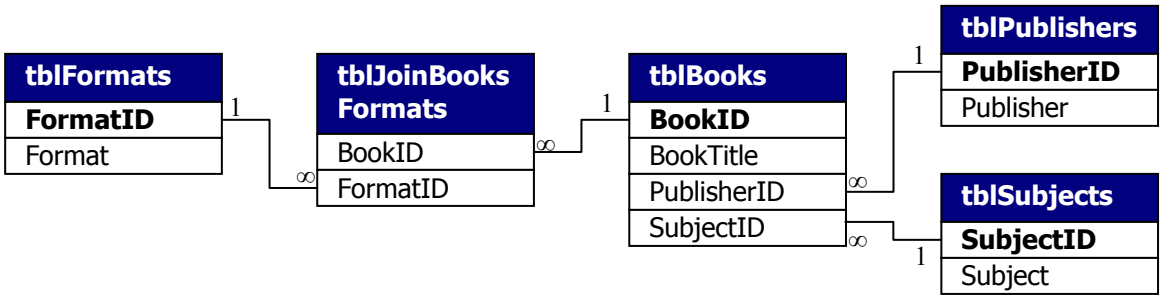
Save and Load Searches in Access VBA

How to allow your users to load and save form states in Access through VBA to provide cross-session saving and retrieval of search or other information.

This article builds on the form-based searching technique developed in the article 'Coding Queries on the Fly from an Access Form Interface' available from www.grbps.com/articles.htm. If you haven't already, you should stop now and read that article before starting this one.

In this article, we will build structures allowing individual searches to be saved and loaded from the search form, allowing the user to assign descriptive names. The saved searches will be available in the current Access session, and in any future session.

Let's recap on the structure of our database. It is used by the owner of a fictitious bookshop specialising in biographies. It's set up like this:



I've populated it with some working data as below and overleaf

tblBooks

BookID	BookTitle	PublisherID	SubjectID
1	Playing the Field	1	1
2	My Drugs Hell	2	3
3	The Downing Street Years	2	2
4	A Game in the Park	2	1
5	Stormy Waters	3	2
6	This Stage of My Life	3	3
7	Cricket and Me	3	1
8	Exeunt Omnes	1	3
9	My Home at Number 10	1	2
10	The Slings and Arrows	1	2

tblPublisher

PublisherID	Publisher
1	Willams and Barnes
2	Scott Laing
3	Marshall Long
4	Templetons

tblFormats

FormatID	Format
1	Hardback
2	Paperback
3	Audio - cassette
4	Audio - CD
5	Ebook

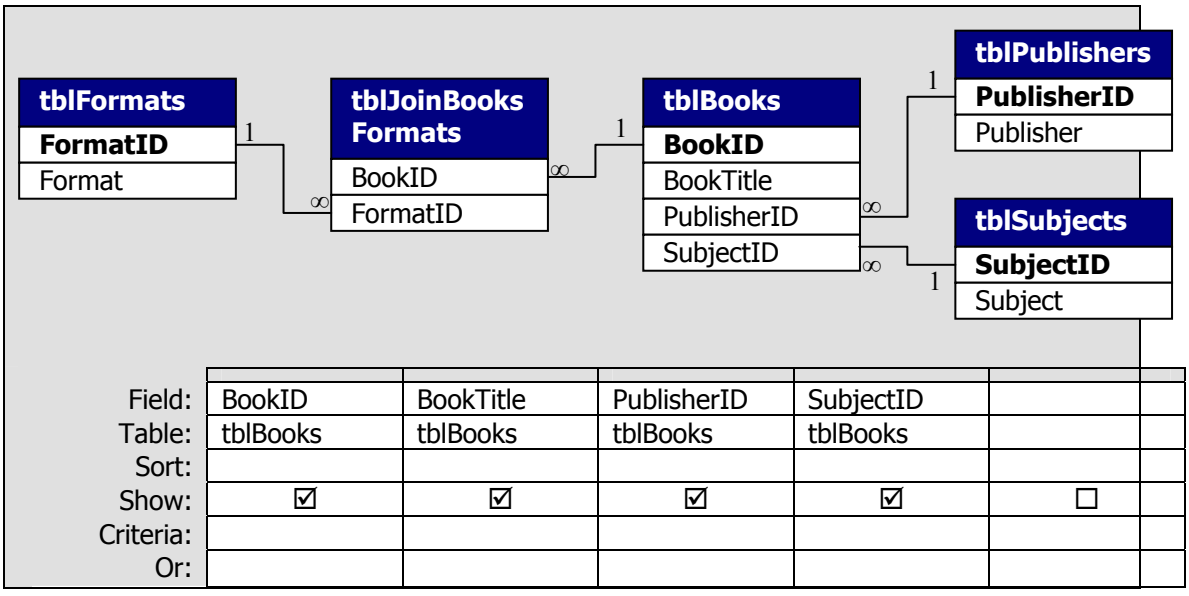
tblSubjects

SubjectID	Subject
1	Sport
2	Politics
3	Entertainment

tblJoinBooksFormats

BookID	FormatID
1	1
1	2
2	2
2	3
2	4
3	2
3	5
4	1
5	2
5	3
6	2
6	4
6	5
7	1
7	2
8	3
8	5
9	1
9	2
10	2

I have a very simple query as below:



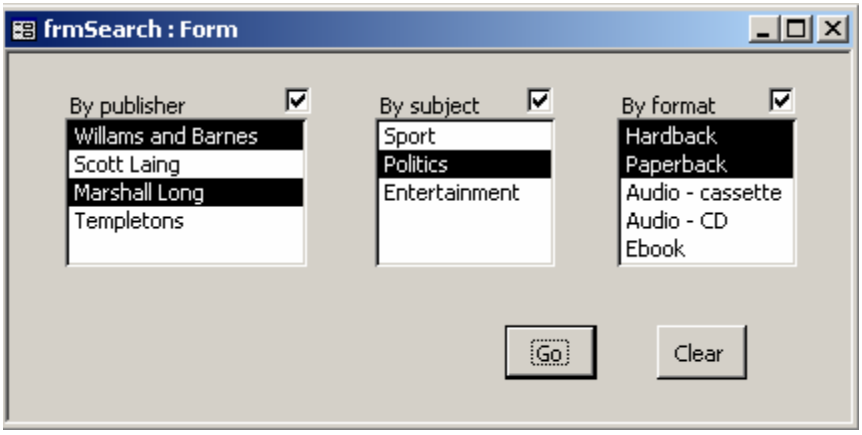
Field:	BookID	BookTitle	PublisherID	SubjectID	
Table:	tblBooks	tblBooks	tblBooks	tblBooks	
Sort:					
Show:	☑	☑	☑	☑	☐
Criteria:					
Or:					

The query properties have 'Unique records' set as 'Yes'.

Here is the resultant SQL which is manipulated in code to generate an infinite number of searches.

```
SELECT DISTINCTROW tblBooks.BookID, tblBooks.BookTitle, tblBooks.PublisherID, tblBooks.SubjectID
FROM tblSubjects INNER JOIN (tblPublisher INNER JOIN (tblFormats INNER JOIN (tblBooks INNER JOIN
tblJoinBooksFormats ON tblBooks.BookID = tblJoinBooksFormats.BookID) ON tblFormats.FormatID =
tblJoinBooksFormats.FormatID) ON tblPublisher.PublisherID = tblBooks.PublisherID) ON tblSubjects.SubjectID =
tblBooks.SubjectID;
```

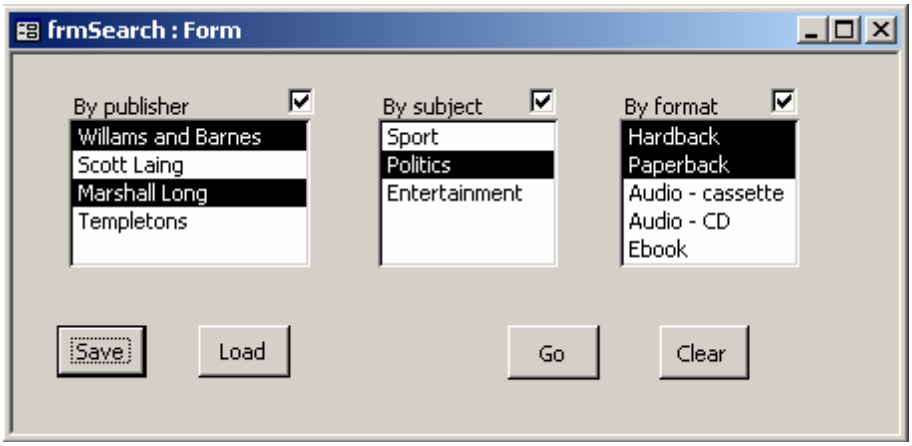
Finally, of course, I have the form which allows the user to specify the search criteria:



With only 3 search fields and no more than 5 elements in each, it's pretty quick and easy to set your search criteria. Imagine this system, though, with 10 or more fields and maybe 20 elements in each. It could take a good few minutes to set the parameters exactly as you want them.

If you regularly had to do the same search, or ones which just differed slightly, you'd like the ability to save the form state and load it back later in exactly the same way. That's the functionality we are now going to add.

First of all, the easy part. Let's put some more buttons on the form, like this:



To be able to save the state of this form, we need to keep a record of:

1. The value (True or False) of each of the check boxes
2. The items selected and not selected in each of the list boxes

It's also a good idea to record the following data, too:

3. A descriptive name for the search
4. The date and time it was saved

The name means that the user has some chance of quickly identifying which saved search is the one he now wants to reload. The date and time is useful, because we can sort the list of saved searches in descending order by date and time, meaning that the most recently saved searches are offered first to the user. We'll be time- and date-stamping each search that is saved for this purpose.

The ideal place to store all this information, of course, is in a table. The only problem is how to store the state of the list boxes. Each of them contains an unlimited number of items. New publishers, subjects or formats could be added at any time. How can I cater for this? It turns out there's an easy way. I'll simply build a string of T and F characters to show whether each of the options is selected ('T') or not selected ('F'). To see how this would work, look at the form pictured on the preceding page. In the Publisher section, there are 4 items in the list box. The first and third are selected, the second and fourth are not. This would translate into a four-character string as below:

```
TFTF
```

The checkbox value is easy, of course, just requiring us to store a true or false value.

Here is the design grid for my new table:

tblSavedSearches: Table			
	Field Name	Data Type	Description
	SearchID	AutoNumber	
	SearchName	Text	
	SearchDate	Date/Time	
	CheckPublisher	Yes/No	
	CheckSubject	Yes/No	
	CheckFormat	Yes/No	
	ListBoxPublisher	Text	
	ListBoxSubject	Text	
	ListBoxFormat	Text	

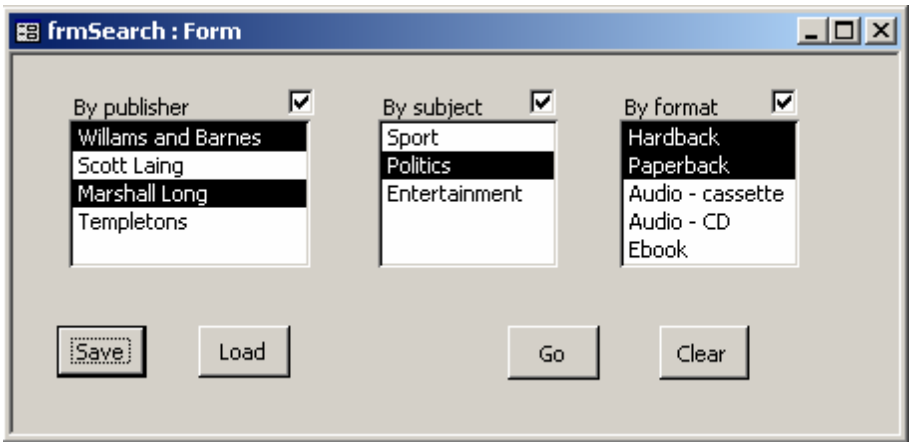
There are a couple of things to note here. Firstly, the text field sizes default to 50 characters. I've left these as they are, catering for up to 50 items per list box. You can increase these for your application if you need to, up to the 255 maximum. Beyond that, you'll need to use a Memo field, but there shouldn't be a problem, although navigating through a list box with so many items may prove tiresome for your users!

The only other adjustment to bear in mind relates to the SearchDate field, where I've set the Default Value property to:

```
=Now()
```

This will constitute my time- and date-stamp, of course.

Take another look at the state of my Search form as it last appeared:



Here's how I want it to translate into my new table:

tblSavedSearches: Table									
	Search ID	Search Name	SearchDate	Check Publisher	Check Subject	Check Format	ListBox Publisher	ListBox Subject	ListBox Format
	1	My test	02/02/2004 10:15:00	☑	☑	☐	TFFT	FFT	FFTFF

So I've assembled my elements. I have a save button on the Search form, but I need to link this to some code. Here's what the code needs to do:

1. Get a name from the user for the search to be saved
2. Find the value of each of the 3 check boxes
3. Convert the list box item selections into strings (see below)
4. Write all these values to a new record in the saved searches table

Step 3 is crying out for a reusable function. Here are the steps which will turn the list box state into a string:

1. Start with an empty string
2. Look at the control
3. Loop through all the items in the control
   If the current item is selected, add 'T' to the string
   If the current item is not selected, add 'F' to the string
   Keep going until all items examined
4. Report the final string

The code, which I have placed in a new module, appears overleaf.

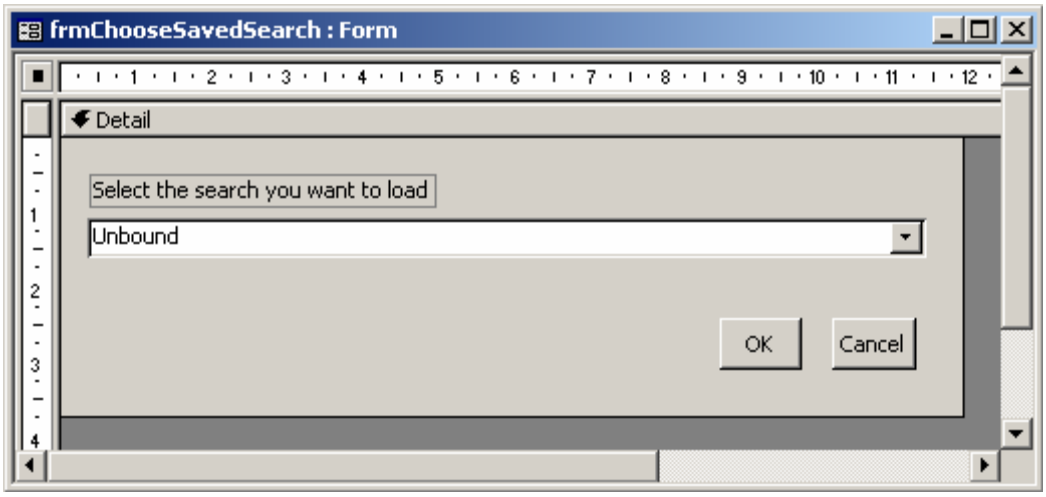
```
Sub SaveSearch()
On Error GoTo Err_SaveSearch

    Dim strNameString As String
    Dim frm As Form
    Dim rs As DAO.Recordset
    Dim booPublisherChk As Boolean
    Dim booSubjectChk As Boolean
    Dim booFormatChk As Boolean
    Dim strPublisherLbo As String
    Dim strSubjectLbo As String
    Dim strFormatLbo As String
    Set frm = Forms("frmSearch")

' get descriptive save name from user
    strNameString = Trim(InputBox("Please enter a descriptive name of this search", _
        "Search name"))

' set check box states
    booPublisherChk = frm.chkPublisher
    booSubjectChk = frm.chkSubject
    booFormatChk = frm.chkFormat

' set list box states using the MakeListSettingsString function
    strPublisherLbo = MakeListBoxSettingsString(frm.lboPublisher)
    strSubjectLbo = MakeListBoxSettingsString(frm.lboSubject)
    strFormatLbo = MakeListBoxSettingsString(frm.lboFormat)

' write to the table
    Set rs = CurrentDb.OpenRecordset("tblSavedSearches")
    With rs
        .AddNew
        !SearchName = strNameString
        !CheckPublisher = booPublisherChk
        !CheckSubject = booSubjectChk
        !CheckFormat = booFormatChk
        !ListBoxPublisher = strPublisherLbo
        !ListBoxSubject = strSubjectLbo
        !ListBoxFormat = strFormatLbo
        .Update
    End With
    rs.Close            ' save resources
    Set rs = Nothing    ' save resources

Exit_SaveSearch:
    Exit Sub

Err_SaveSearch:
    MsgBox Err.Description
    Resume Exit_SaveSearch

End Sub
```

```
Function MakeListBoxSettingsString(conControl As Control)
    Dim strCritString As String
    Dim intCurrentRow As Integer

    strCritString = ""
' Loop through the listbox control and see whether each item is selected or not
    For intCurrentRow = 0 To conControl.ListCount - 1
        If conControl.Selected(intCurrentRow) Then
            strCritString = strCritString & "T"       ' item selected
        Else: strCritString = strCritString & "F"   ' item not selected
        End If
    Next intCurrentRow

' Return the string to the main routine
    MakeListBoxSettingsString = strCritString

End Function
```

It should be pretty straightforward to follow, particularly when compared against the program flow order detailed above. Note that I never write to the SearchDate field, allowing it to automatically populate with the default value.

You might want to play at this point and create a few saved searches.

Now that we have the ability to save searches, we can move to building the loading routines. The first thing I'll need is a new form to choose which saved search to use. When saving, I didn't need to do this; I could get the text string through a simple input box from VBA. This time, though, I want to present the user a choice with a combo box and a cancel button. Consequently, I've built a new form called 'frmChooseSavedSearch':



The combo box is called 'cboSearchName' and its Row Source is:

```
SELECT tblSavedSearches.SearchID, tblSavedSearches.SearchName,
tblSavedSearches.SearchDate FROM tblSavedSearches ORDER BY
tblSavedSearches.SearchDate DESC;
```

The Bound column is column 1, and column widths are set to zero for the SearchID and SearchDate fields. This allows the combo box to show just the search names, but for them to appear in descending time and date order. It returns the SearchID of the selected item.

So what I need to do is write code to take the SearchID returned and pass all the related records back to the Search form. There are two ways to do this. I could simply have hidden fields on this form to carry all the other fields, or I could read the record direct from a Recordset object. Either way, I'll need VBA to convert the strings back to list box settings and write all the values back to the search form. The hidden fields solution strikes me as messy, so I'm going to use the Recordset method.

Here are the steps I need my code to undertake:

1. Find the full record that goes with the selected SearchID
2. Set the check box values on the Search form
3. Convert the strings into list box settings on the Search form
4. Close the frmChooseSavedSearch form

At this point you may be wondering why I'm setting the form state rather than just cutting out the middle man and performing the search. Well, one reason is that it allows the user to check that the search is going to do what they think it is. Another is that they might want to load a particular saved search and make certain adjustments as a shortcut rather than starting with a blank search form. This method of simply setting the search parameters gives your user more flexibility.

It is the conversion at Step 3 which again cries out for a reusable subroutine. Here is the code, including the Step 3 subroutine. It is in a module with the earlier code and is called by the Click event of the OK button on the frmChooseSavedSearch form.

```
Sub LoadSavedSearch()
On Error GoTo Err_LoadSavedSearch

    Dim intSearchID As Integer
    Dim frm As Form
    Dim conControl As Control
    Dim rs As DAO.Recordset
    Dim strSQL As String

' Open the Search form if it isn't already
    DoCmd.OpenForm "frmSearch"
    Set frm = Forms!frmSearch

' Read the SearchID value selected by the user
    intSearchID = Forms!frmChooseSavedSearch.cboSearchName.Column(0)

' Load the single record related to the slected Search ID as a Recordset
    strSQL = "SELECT tblSavedSearches.* " & _
        "FROM tblSavedSearches " & _
        "WHERE tblSavedSearches.[SearchID] = " & intSearchID
    Set rs = CurrentDb.OpenRecordset(strSQL)

    rs.MoveFirst        ' select the record

' set check box values
    frm.chkPublisher = rs!CheckPublisher
    frm.chkSubject = rs!CheckSubject
    frm.chkFormat = rs!CheckFormat

' set list box values using the ResetlistBoxFromString function
    ResetListBoxFromString frm.lboPublisher, rs!ListBoxPublisher
    ResetListBoxFromString frm.lboSubject, rs!ListBoxSubject
    ResetListBoxFromString frm.lboFormat, rs!ListBoxFormat

    rs.Close                ' save resources
    Set rs = Nothing        ' save resources

Exit_LoadSavedSearch:
    DoCmd.Close acForm, "frmChooseSavedSearch"      'close dialog form
    Exit Sub

Err_LoadSavedSearch:
    MsgBox Err.Description
    Resume Exit_LoadSavedSearch

End Sub
```

```
Sub ResetListBoxFromString(conControl As Control, strSetting As String)

    Dim intCount As Integer

' Loop through string characters and set selection status of list box items
    For intCount = 1 To Len(strSetting)
        If Mid(strSetting, intCount, 1) = "T" Then
            conControl.Selected(intCount - 1) = True
        Else
            conControl.Selected(intCount - 1) = False
        End If
    Next intCount

End Sub
```

And that's about it, really. The Cancel button on the frmChooseSavedSearch form just needs a single line of code:

```
DoCmd.Close
```

Well, we're done. We started with a search system which rewrote queries on the fly from a forms interface. We then build an approach to store information on the form's state across sessions and then extract that stored information to recreate the form's state at a later time. For our user, it's just as intuitive as saving and loading a file from the hard drive and the technique has applications in all sorts of Access projects.