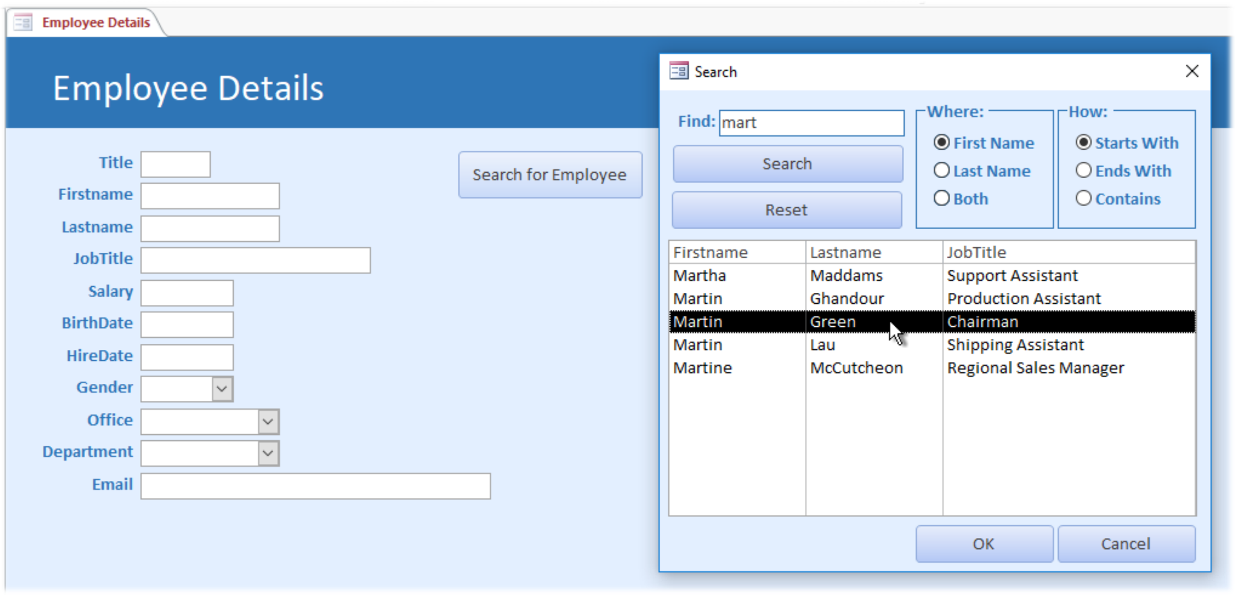
Access Forms Masterclass 4
A Pop-Up Search Tool

Published: 10 September 2018
Author: Martin Green
Screenshots: Access 2016, Windows 10
For Access Versions: 2007, 2010, 2013, 2016

A Custom Search Tool for Your Access Database

I have added this custom search tool to many of the databases I have built. Access forms include simple built-in search tools but, whilst useful, they are limited to simple text searches on individual fields, but how many of your users know of the existence of the options on the right-click context menu?

My search tool consists of a pop-up dialog box (*Fig. 1*) that offers the user the ability to search for a specific record, through a choice of fields and in several different ways.



Find: mart

Where: [✓ First Name] Last Name Both

How: [✓ Starts With] Ends With Contains

Firstname	Lastname	JobTitle
Martha	Maddams	Support Assistant
Martin	Ghandour	Production Assistant
Martin	Green	Chairman
Martin	Lau	Shipping Assistant
Martine	McCutcheon	Regional Sales Manager

Fig. 1 A pop-up search tool.

The search tool is usually opened from a button on a form. It opens to display a complete list of records. The user enters a search string into the tool's *Find* text box then chooses which field or fields they want to search, and how they want the search performed. The search string is not case sensitive. It can be a single letter or string of letters, and it can include an asterisk (*) to represent any one or more characters, or question mark (?) to represent any single character.

To perform the search the user simply clicks the *Search* button and the search tool's list changes to show the results. They can restore the original list by clicking the *Reset* button or perform another search without needing to refresh the list.

If they fail to find the item they are looking for they can close the search tool by clicking the *Cancel* button but if they have found the required item they can select it in the list then click the *OK* button to close the dialog box and display the chosen record in their form.

In this Masterclass I am going to lead you through the steps of building a tool that will allow the user to search for a specific record within their data. In this example we are looking for a person by name by entering a string of characters and choosing whether to search for a *First Name* or *Last Name*. You should be able to apply the same technique to search any kind of data, the only proviso being that the data you are searching comes from a table that has a *Primary Key* field. You will see the importance of this later.

You probably already know how to build a form in Access but, in case you don't, I'm going to take you through all the necessary steps. The are several tasks to perform but you can pause at any time. Just remember to save your work when you do.

Step 1: Build the Search Form

The search tool takes the form of a custom dialog box. A dialog box is basically an Access form that is not linked to an underlying recordset and that has certain settings applied so that, usually, it floats in front of the user's workspace.

1.1 Open a New Blank Form

Click the **Form Design** button (*Fig. 2*) on the **Create** tab of the Access ribbon to create a new, empty form.

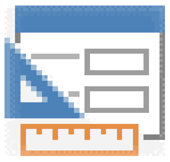


Fig. 2 The Form Design tool.

I'm a firm believer in regularly saving my work in case something goes wrong so now is a good time to start. Click the **Save** button in the upper left corner of the Access window or press **[Ctrl]+S** to bring up the *Save As* dialog and give your new form a name. For this exercise I am using the name *frmSearch*. From now on you can save your changes simply by repeating the *Save* command.

So far, the form consists solely of a *Detail* area (the form's background) which you can colour and resize as necessary. I like to colour the form's background to suit the colour scheme of my database at this stage. To do this, if the *Property Sheet* isn't already open, **right-click** on the detail area and choose **Properties** from the context menu. Go to the **Format** tab and choose a suitable colour for the *Back Color* property. I have chosen *Access Theme 2* (*Fig. 3*) a pale blue colour which is one of the presets on the drop-down list. Alternatively click the build (**[...]**) button to apply any colour of your choice.

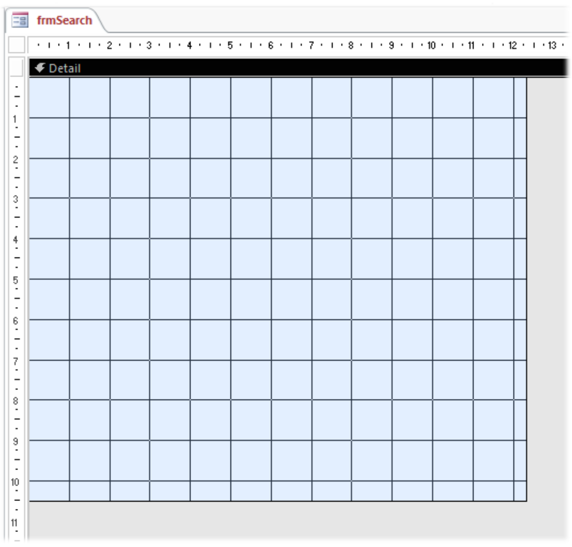


Fig. 3 A new empty form.

1.2 Add a Text Box

The first thing to add to the form is a Text Box into which the user will enter a search string. They will be able to enter a single letter, a string of letters or an entire name. Click the **Text Box** button (*Fig. 4*) on the **Design** tab of the ribbon to activate the Text Box tool then click on the form, somewhere near the centre of in the upper part of the Detail area to place a new Text Box and its label on the form.

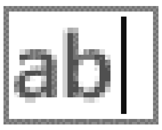


Fig. 4 The Text Box tool.

Enter some suitable text into the label (I used *Find:*) then move the text box and its label into the upper left part of the detail area (*Fig. 5*).

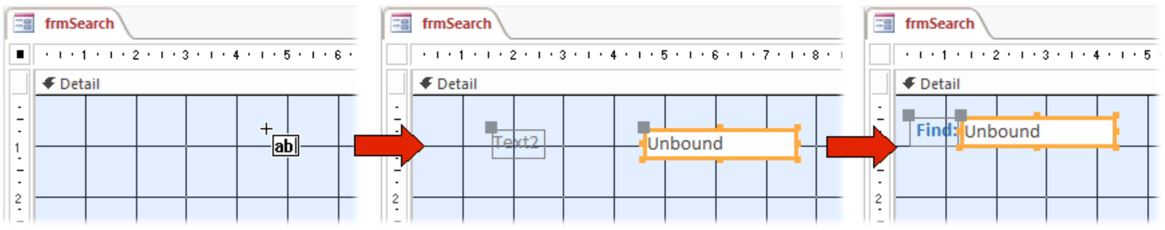


Fig. 5 Position the Text Box and enter suitable text to the label.

TIP: To resize a selected control (items on a form are called *controls*) drag one of the dots on the control's border. To move a control, drag the border itself (not a dot). When a control has an accompanying label the label also moves when you move the control. To move a control or its label independently drag it by the large grey dot in its upper left corner. You can also use the keyboard to move and resize a control: click on the control to select it then use the **[Arrow]** keys to move an item. Hold down **[Shift]** as you do so to resize the item. Hold down **[Ctrl]** whilst moving or resizing for fine-tuning.

1.3 Add the Option Groups

In this example the user will be searching through a list of names. They will be offered two sets of choices for searching:

- **Where:** they can choose to search in the *Firstname* field, in the *Lastname* field or in *both* fields.
- **How:** they can look for a name that *Starts With*, *Ends With* or *Contains* the string that they entered in the text box.

To achieve this, we are going to add two *Option Groups*. An Option Group consists of a frame with a label. *Option Buttons* will be placed inside each frame, each button representing one of the choices. When an Option Button is used alone on a form it has a value of *True* or *False* but when Option Buttons are placed inside an Option Group frame they each have a numerical value and only one can be selected at a time. The value of the Option Group becomes that of the selected Option Button.

Start by using the **Option Group** tool *(Fig. 6 left*) to draw a rectangle on the form, then use the **Option Button** tool (*Fig. 6 right*) to place three option buttons inside the rectangle.

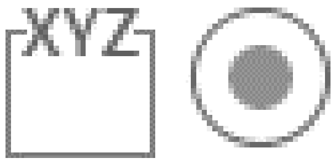


Fig. 6 The Option Group (left) and Option Button (right) tools.

Add a suitable title to the Option Group's label (e.g. *Where:*) and to each option button (*First Name, Last Name, Both*). In this example (*Fig. 7*) I have applied formatting to match the design and colour scheme of my form, making the text bold and blue, and changing the Special Effect property of the Option Group border from *Etched* to *Flat*, also colouring the border blue.

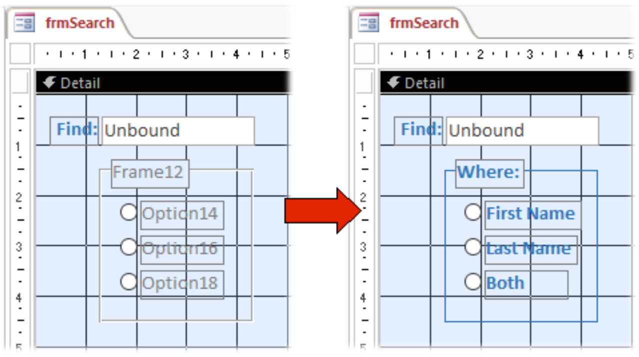


Fig. 7 Name and format parts of the Option Group.

You need a second Option Group so repeat the process or, more easily, simply select your first Option Group by using the mouse to drag a rectangle enclosing the entire Option Group, and pressing **[Ctrl]+C** then **[Ctrl]+V** to Copy and Paste, making an exact copy of it.

Change the labels of the second Option Group, for example *How:* for the Option Group label and *Starts With, Ends With* and *Contains* for the Option Buttons. Format your controls as you wish and position them as you prefer on the form, then check out your design by switching the form into **Form View** (*Fig. 8*) using the **Views** button on the **Home** tab of the ribbon.

TIP: Now would be a good time to save your work. Remember to save your changes regularly.

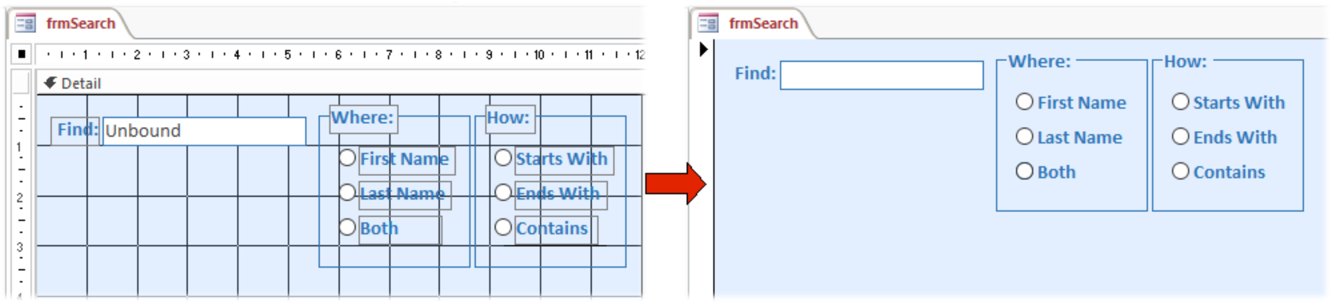


Fig. 8 Check the form's appearance in Form View.

You might notice that in my example I have chosen to place the Option Groups to the right of the search Text Box and have also widened the Text Box and coloured its border blue to match my form's colour scheme.

Now we can set the properties of the controls we have placed on the form so far. Select the Text Box, go to the **Other** tab of the **Property Sheet** and change its **Name** property to *txtSearch*. Select the Frame of the first Option Group (not the whole group) and change its name to *grpWhere*. Name the second Option Group *grpHow*.

It isn't necessary to name the individual Option Buttons because we won't be referring to them in the code but you need to check the *Option Value* of each button on the **Data** tab of the **Property Sheet**. Access numbers the buttons' value consecutively as you add them to the group but it is best to check anyway. They should be *1, 2* and *3* respectively in each group.

TIP: Sometimes I create a set of Option Buttons by placing one in the frame then copying it and pasting as many times as I need buttons. If you do this remember to change their Option Values because, being copies of the first one, they will all have the same value unless you change it.

Finally, set the *Default Value* property of each Option Group to *1* to make sure that one of the buttons is already selected when the search tool opens. You won't notice any difference by simply switching to Form View but if you close, save and reopen the form you will see that the first option button of each group is now selected. This avoids the possibility of the user trying to perform a search with no option selected.

1.4 Add the List Box

The next task is to add a List Box to display the results of the search. Use the **List Box** tool to draw a large rectangle on the form (*Fig. 9*). Don't worry too much about the exact size of the box because you can adjust that later. It doesn't need its label so select the Label and remove it by pressing the **[Delete]** key. Change the **Name** property of the List Box to *lstSearch*.

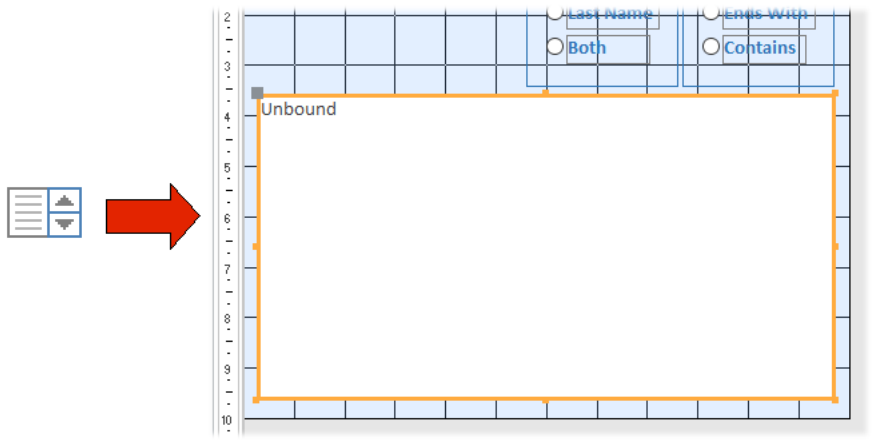
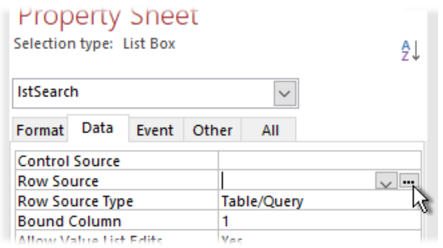


Fig. 9 Draw a large List Box on the form.

Your List Box can show as much information as you require. In this example the user is searching the data in the *tblEmployees* table so I want to display the *Firstname, Lastname* and *JobTitle* fields. In addition the List Box list must include the Primary Key field which, in this example, is the Autonumber field *EmployeeID*.

To populate the list box select it then go to the **Data** tab of the **Property Sheet**, click in the **Row Source** box then click the **Build** button (**[...]**) to open the Query Builder (*Fig. 10*).



Selection type: List Box

lstSearch

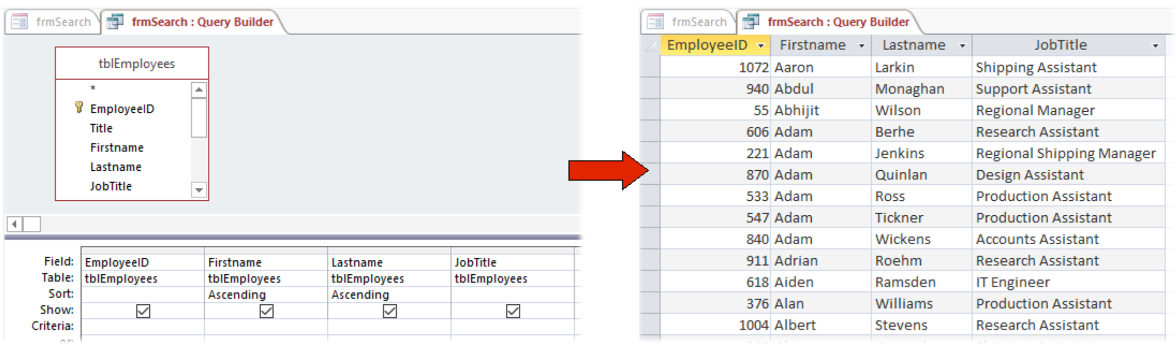
Control Source	
Row Source	
Row Source Type	Table/Query
Bound Column	1

Fig. 10 Open the Row Source Query Builder.

Just as you would when creating a regular Select query, add the fields you would like the user to see in the search results List Box. I have chosen *EmployeeID* (the Primark Key field), *Firstname, Lastname* and *JobTitle*. It is important that you include the Primary Key field in your query, and make sure that it is the first field returned by the query. I also chose to sort my query by *Firstname* then *Lastname*. You can do whatever you prefer.

You can run the query to check that it is working as you wish (*Fig. 11*). This is how the user will see the data in the List Box except for the Primary Key field which will be present but hidden. When you are satisfied with your query close the Query Builder window and confirm that you want to use the query as the List Box Row Source when asked. You will see the query's SQL statement written in the Row Source box on the Property Sheet.

Tip: If you are familiar with SQL you can type a suitable SQL statement directly into the Row Source box.



Field:	EmployeeID	Firstname	Lastname	JobTitle
Table:	tblEmployees	tblEmployees	tblEmployees	tblEmployees
Sort:		Ascending	Ascending	
Show:	✓	✓	✓	✓
Criteria:				

EmployeeID	Firstname	Lastname	JobTitle
1072	Aaron	Larkin	Shipping Assistant
940	Abdul	Monaghan	Support Assistant
55	Abhijit	Wilson	Regional Manager
606	Adam	Berhe	Research Assistant
221	Adam	Jenkins	Regional Shipping Manager
870	Adam	Quinlan	Design Assistant
533	Adam	Ross	Production Assistant
547	Adam	Tickner	Production Assistant
840	Adam	Wickens	Accounts Assistant
911	Adrian	Roehm	Research Assistant
618	Aiden	Ramsden	IT Engineer
376	Alan	Williams	Production Assistant
1004	Albert	Stevens	Research Assistant

Fig. 11 Use the Query Builder to create a SQL statement for the List Box Row Source property.

If you were to check the result in Form View now you would see that the List Box is showing only the first field that was returned by the query (*EmployeeID*). The next step is to configure the List Box so that it displays all the required fields correctly.

You will see on the Property Sheet that the *Bound Column* property of the List Box is column *1*. This represents the first field returned by the query which, correctly, is our Primary Key field. Being the *Bound Column* means that, when an item is selected, the value of this field is the value that the List Box assumes.

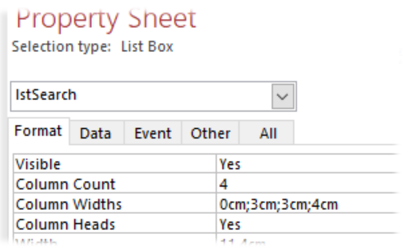
> NOTE: List Boxes can be set to allow the user to make a single selection (the default setting and that used here) or multiple selections. If you are interested in using a List Box for multiple selections read my tutorial *Making Sense of List Boxes* at http://www.fontstuff.com/access/acctut11.htm.

With the List Box selected, go to the **Format** tab of the **Property Sheet** and make the following settings (*Table 1*):

Table 1: List Box Properties

Property	Tab	Value
Column Count	Format	4
Column Widths	Format	0cm;3cm;3cm;4cm
Column Heads	Format	Yes (optional)
Bound Column	Data	1

The exact settings will depend upon your data and the width of your List Box. Check the result in Form View. There is usually an amount of trial-and-error involved in getting it right.

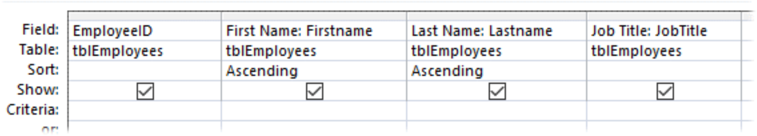

Property Sheet
Selection type: List Box

lstSearch

Visible	Yes
Column Count	4
Column Widths	0cm;3cm;3cm;4cm
Column Heads	Yes

Fig. 12 Set the List Box properties.

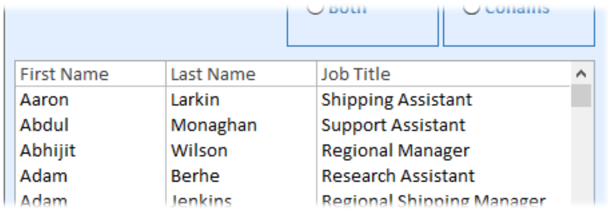
I have chosen to show Column Heads (*Fig. 12*). Access uses the field names for this but, if your field names are not appropriate or sufficiently meaningful you can choose not to show them. Alternatively, you have a couple of choices: go to the design view of the source table and set the *Caption* property of the field to something more appropriate. If you prefer not to get involved with the table design you can modify your Row Source query to add a custom caption for each field. To do this type your chosen caption, followed by a colon (**:**), in front of the field name in the grid as shown here (*Fig. 13*).



Field:	EmployeeID	First Name: Firstname	Last Name: Lastname	Job Title: JobTitle
Table:	tblEmployees	tblEmployees	tblEmployees	tblEmployees
Sort:		Ascending	Ascending	
Show:	✓	✓	✓	✓
Criteria:				

Fig. 13 You can customize the field titles in the Row Source query.

When you are satisfied with the layout of the data in your List Box (*Fig. 14*) save your changes.



First Name	Last Name	Job Title
Aaron	Larkin	Shipping Assistant
Abdul	Monaghan	Support Assistant
Abhijit	Wilson	Regional Manager
Adam	Berhe	Research Assistant
Adam	Jenkins	Regional Shipping Manager

Fig. 14 The completed List Box.

1.5 Add the Command Buttons

You need four command buttons:

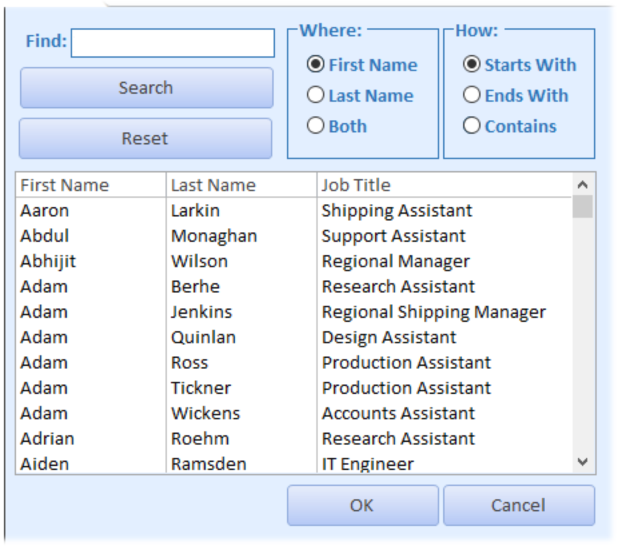
- **Search** The user will click this button after entering their search criteria to perform the search and display the results in the List Box.
- **Reset** Clicking this button will return the List Box list to its original state (i.e. showing the full list). Although it will not be necessary to reset the list before running another search, it might be convenient for the user to do so. You could omit this button if you wish.
- **OK** Having found and selected a suitable list entry the user will click this button to close the search tool and return to their original form and view the chosen record.
- **Cancel** This button will close the search tool without performing any other task.

Add four command buttons to your form and give an appropriate name and caption to each one using the *Name* and *Caption* properties on the Property Sheet. In this example I have used **Captions** and *Names* as follows:

Search, *cmdSearch*; **Reset**, *cmdReset*; **OK**, *cmdOK*; **Cancel**, *cmdCancel*

TIP: You can write whatever you like for the caption but keep the name simple and relevant. It can be very frustrating when writing code trying to remember what you called something when cryptic names are used.

You can lay your form out in any way that pleases you. Here's how mine looks after a few adjustments (*Fig. 15*):



Where: [✓ First Name] Last Name Both

How: [✓ Starts With] Ends With Contains

First Name	Last Name	Job Title
Aaron	Larkin	Shipping Assistant
Abdul	Monaghan	Support Assistant
Abhijit	Wilson	Regional Manager
Adam	Berhe	Research Assistant
Adam	Jenkins	Regional Shipping Manager
Adam	Quinlan	Design Assistant
Adam	Ross	Production Assistant
Adam	Tickner	Production Assistant
Adam	Wickens	Accounts Assistant
Adrian	Roehm	Research Assistant
Aiden	Ramsden	IT Engineer

Fig. 15 The finished layout of the Search tool.

Next, we need to add some VBA code to power the command buttons.

Step 2: Write Code to Power the Command Buttons

Having equipped the form with all the necessary controls the next task is to create the VBA code that will make it work. Each of the command buttons requires a macro (called an *Event Procedure*) that will run when the user clicks the button.

2.1 The Cancel Button

Let's start with the simplest one. All that this button's code needs to do is to close the form. Select the **Cancel** button and go to the **Event** tab of the Properties Sheet. We need to have a macro run when the user clicks this button so click in the **On Click** text box then click the **Build** button to display the *Choose Builder* dialog. Choose **Code Builder** (*Fig. 16*) then click **OK.**

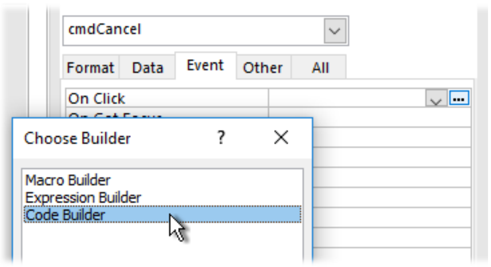
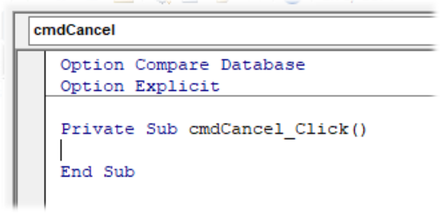


Fig. 16 Choose the Code Builder.

TIP: A quicker way to do this is to double-click in the Event text box (the entry **[Event Procedure]** will appear in the box) then click the build button.

This takes you to the Visual Basic Editor with an empty Click event macro already written and your cursor placed inside (*Fig. 17*) ready for you to enter the code.


cmdCancel

```
Option Compare Database
Option Explicit

Private Sub cmdCancel_Click()

End Sub
```

Fig. 17 The Visual Basic Editor has created an empty Click event macro.

TIP: You can save time typing by selecting and copying the code you see here and pasting it directly into the Visual Basic Editor.

Enter the code statement as follows (*Listing 1*):

Listing 1:

```
Private Sub cmdCancel_Click()
    On Error Resume Next
    DoCmd.Close acForm, "frmSearch"
End Sub
```

How it Works:

This simple code statement closes the named form which, in this case, is the current form.

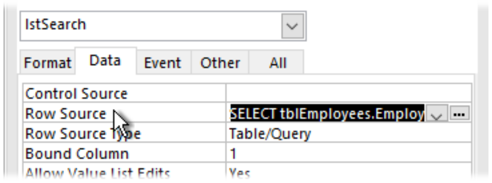
NOTE: You will notice that, as you type, the Visual Basic Editor often provides a list of entries, relevant to what you have typed. You can carry on typing but, if you prefer, you can select an item from the list by double-clicking it. When appropriate, the Visual Basic Editor also displays a helper showing what parameters are required, the entry in bold being the item it expects next.

You will see that I have included a simple error handler at the start of the macro. It is good practice to always include an error handler in all your macros, just in case something goes wrong.

2.2 The Reset Button

The purpose of the Reset button is to return the List Box to its original state by returning its Row Source to its original value. This can be done easily with code.

Start by making a copy of the Row Source property of the List box. To do this select the List Box then go to the **Data** tab of the Property Sheet and click the label of the *Row Source* property. This highlights everything in the *Row Source* text box (*Fig. 18*). Now either press **[Ctrl]+C** or **right-click** on the highlighted text and choose **Copy** to copy it.



Control Source	
Row Source	SELECT tblEmployees.Employ
Row Source Type	Table/Query
Bound Column	1
Allow Value List Edits	Yes

Fig. 18 Click the property label to highlight the text.

Create an Event Procedure for the *On Click* event of the *Reset* button exactly as you did in the previous step and enter its code as shown below (*Listing 2*). I suggest you read the whole of this section before you start typing.

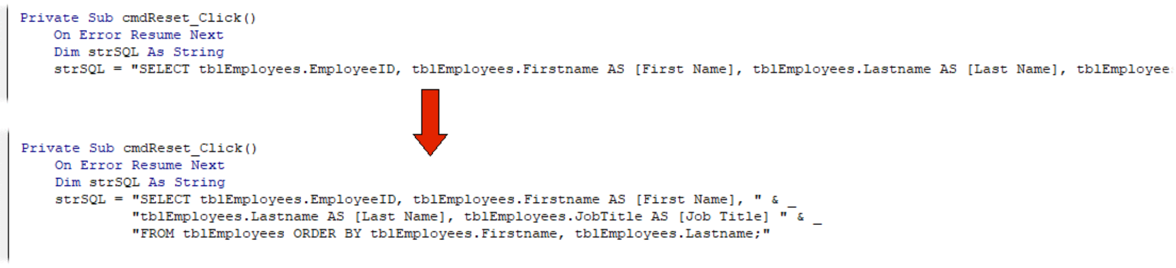
Listing 2:

```
Private Sub cmdReset_Click()
    On Error Resume Next
    Dim strSQL As String
    strSQL = "SELECT tblEmployees.EmployeeID, tblEmployees.Firstname, " & _
             "tblEmployees.Lastname, tblEmployees.JobTitle " & _
             "FROM tblEmployees " & _
             "ORDER BY tblEmployees.Firstname, tblEmployees.Lastname;"
    With Me.lstSearch
        .RowSource = strSQL
        .Requery
    End With
    Me.grpHow.Value = 1
    Me.grpWhere.Value = 1
End Sub
```

How it Works:

Following the Error Handler (which must always be at the very top of your macro) there is a statement declaring the variable *strSQL*. This variable is simply a container to temporarily hold the text of the query that forms the Row Source of the list box. The process of declaring a variable warns the code engine that a particular piece of text (*strSQL*) is being used to represent that variable and also what sort of data it will contain (here a text string).

The next statement places a value into the variable. To do this, after typing the equals sign, type an opening quote mark then simply paste the SQL statement you copied from the Row Source property earlier. Remember to type a closing quote mark at the end of the SQL. When you paste your SQL statement you will see that is arrives as a long line of text running off the right side of the window. This is OK but I prefer to enter it in a way that I can see the whole thing when viewing my code (*Fig. 19*). Access doesn't care as long as it is done correctly!



```
Private Sub cmdReset_Click()
    On Error Resume Next
    Dim strSQL As String
    strSQL = "SELECT tblEmployees.EmployeeID, tblEmployees.Firstname AS [First Name], tblEmployees.Lastname AS [Last Name], tblEmployee
```

```
Private Sub cmdReset_Click()
    On Error Resume Next
    Dim strSQL As String
    strSQL = "SELECT tblEmployees.EmployeeID, tblEmployees.Firstname AS [First Name], " & _
             "tblEmployees.Lastname AS [Last Name], tblEmployees.JobTitle AS [Job Title] " & _
             "FROM tblEmployees ORDER BY tblEmployees.Firstname, tblEmployees.Lastname;"
```

Fig. 19 Breaking a very long code statement to fit on the screen.

When you break lines of code like this you must do it correctly. Because we are breaking a text string each part must be surrounded by quote marks and an ampersand (**&**) used to tell Access that the next string of text continues directly from the current one. Following the ampersand is the Line Break Character (a space followed by an underscore) which tells Access that the code statement continues on the next line.

The code continues with two commands. The first command applies the SQL statement (in the form of the *strSQL* variable) to the *RowSource* property of the List Box. This alone does not cause the contents of the List Box to change, which is why we need the second command telling it to *Requery* which makes it rebuild its list.

Finally, a pair of code statements return the two option groups to their default settings by making the value of each = 1.

Now would be a good time to Compile and Save your code. In the Visual Basic Editor open the **Debug** menu and choose **Compile...** (*Fig. 20*) then click the **Save** button.

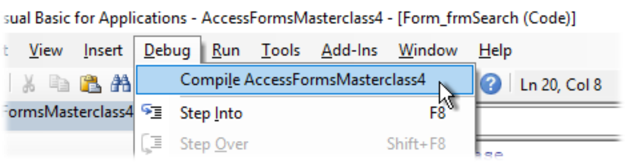

sual Basic for Applications - AccessFormsMasterclass4 - [Form_frmSearch (Code)]
View Insert Debug Run Tools Add-Ins Window Help
Compile AccessFormsMasterclass4
Step Into F8
Step Over Shift+F8
Ln 20, Col 8
ormsMasterclass4

Fig. 20 Regularly Compile and Save your code.

TIP: It is good practice to regularly Compile and Save your code as you are working. Compiling checks the integrity of your code, picking up any errors that were not highlighted as you were typing. Taking the opportunity to Save at the same time helps to avoid losing valuable work in the event that the program crashes when you try to run a faulty macro.

2.3 The Search Button

The Row Source property of the List Box takes the form of a SQL Statement that we generated with the aid of the Query Builder. The task of the code behind the Search button is to replace this SQL statement with one that generates a recordset that satisfies the user's search requirements.

The code must read the Search string entered by the user as well as their choices of Where and How to search. This information will be used to construct a suitable SQL statement which will define the Row Source of the list box.

The way I approach this task varies according to the number and nature of the choices that I am offering the user. In this example I am using two Case Statements to interpret the choices that the user makes in the two Option Groups. A case statement works like an IF Statement in that it can make a decision according to the information it is given. When there are more than a couple of options I find that a Case Statement is much easier to write and understand than a complicated If Statement.

The process of building the SQL begins with a Case Statement that looks at the choice the user made in the **Where** option group. It does this by examining the value of that group i.e. *1 = Firstname*, *2 = Lastname*, *3 = Both* (*Firstname* or *Lastname*).

For each of these choices there are another three possibilities as chosen by the user in the **How** option group. For each of the three cases in my first Case Statement there is another "nested" Case Statement examining the value of the second option group i.e. *1 = Starts With, 2 = Ends With, 3 = Contains*. The completed macro (*Listing 3*) is shown below.

Listing 3:

```
Private Sub cmdSearch_Click()
   On Error Resume Next
   Dim strSearch As String
   Dim strSQL As String
   strSearch = Me.txtSearch.Value
   Select Case Me.grpWhere.Value
      Case 1 'Firstname
         Select Case Me.grpHow.Value
            Case 1 'Starts With
               strSearch = "[Firstname] Like " & Chr(34) & strSearch & "*" & Chr(34)
            Case 2 'Ends With
               strSearch = "[Firstname] Like " & Chr(34) & "*" & strSearch & Chr(34)
            Case 3 'Contains
               strSearch = "[Firstname] Like " & Chr(34) & "*" & strSearch & "*" & Chr(34)
         End Select
      Case 2 'Lastname
         Select Case Me.grpHow.Value
            Case 1 'Starts With
               strSearch = "[Lastname] Like " & Chr(34) & strSearch & "*" & Chr(34)
            Case 2 'Ends With
               strSearch = "[Lastname] Like " & Chr(34) & "*" & strSearch & Chr(34)
            Case 3 'Contains
               strSearch = "[Lastname] Like " & Chr(34) & "*" & strSearch & "*" & Chr(34)
         End Select
      Case 3 'Both
         Select Case Me.grpHow.Value
.           Case 1 'Starts With
               strSearch = "[Firstname] Like " & Chr(34) & strSearch & "*" & Chr(34) & _
                           " OR [Lastname] Like " & Chr(34) & strSearch & "*" & Chr(34)
            Case 2 'Ends With
               strSearch = "[Firstname] Like " & Chr(34) & "*" & strSearch & Chr(34) & _
                           " OR [Lastname] Like " & Chr(34) & "*" & strSearch & Chr(34)
            Case 3 'Contains
               strSearch = "[Firstname] Like " & Chr(34) & "*" & strSearch & "*" & Chr(34) & _
                           " OR [Lastname] Like " & Chr(34) & "*" & strSearch & "*" & Chr(34)
         End Select
   End Select
   strSQL = "SELECT tblEmployees.EmployeeID, tblEmployees.Firstname, " & _
            "tblEmployees.Lastname, tblEmployees.JobTitle " & _
            "FROM tblEmployees " & _
            "WHERE " & strSearch & " " & _
            "ORDER BY tblEmployees.Firstname, tblEmployees.Lastname;"
   With Me.lstSearch
       .RowSource = strSQL
       .Requery
   End With
End Sub
```

How it Works:

After the error handler two variables are declared, *strSearch* to hold the SQL string that will define the WHERE clause of the Row Source, and *strSQL* which will hold the final SQL string that will make up the Row Source itself. Next comes a statement that places the user's entry from the Find text box into the *strSearch* variable.

The first Case Statement checks the user's choice from the Where option group (*1, 2* or *3*) and moves to the relevant part of the Case Statement where it encounters a second Case Statement. It uses this second Case Statement to check the user's choice from the How option group (again *1, 2* or *3*) and moves to the relevant part of this case statement. Here it finds a text string that describes in SQL the user's choices from the Find text box, the Where option group and the How Option Group.

> NOTE: You will see that in the VBA that creates the SQL I have used Chr(34) a number of times. Chr(34) uses the Chr() function to represent the quote mark (") which has the ASCII value 34. It is used here because in VBA a string must be enclosed in quotes, but if that string itself includes quotes an anomaly will occur e.g. Like "M*" becomes "Like "M*"". Some developers deal with this by nesting different kinds of quotes e.g. "Like 'M*'" but I have found that this can sometimes be misinterpreted by the database engine so I always use Chr(34) instead. It takes a little extra care but will always work correctly.

For example, if the user enters *gr* in the Find text box then chooses *Last Name* and *Starts With* the code statement:

```
strSearch = "[Lastname] Like " & Chr(34) & strSearch & "*" & Chr(34)
```

generates the SQL:

```
[Lastname] Like "gr*"
```

Which is then added to the WHERE clause of the SQL statement that forms the Row Source of the List Box as:

```
WHERE [Lastname] Like "gr*"
```

TIP: If you want to find the ASCII value of a particular character in VBA you can use the Asc() function to do so. In the Visual Basic Editor open the Immediate window (Keys: **[Ctrl]+G** or **View** > **Immediate Window**) and type a question mark followed by the Asc() function containing the character in question enclosed in quotes e.g ?Asc("@"). Press **[Enter]** and the ASCII value of the character will be returned (in this case 64).

Finally, the search string held in the strSearch variable is added to the remaining SQL that will form the Row Source of the List Box. To complete the process the completed SQL statement is applied to the Row Source property of the List Box and the Requery command tells it to it refresh its list.

2.4 The OK Button

Having searched the database and found the item they were looking for, our user will select the item in the List Box and click the OK button to take them to the relevant record.

In this example the records will be displayed on a form named *frmEmployees*. The code must identify which record the user has chosen from the list then instruct the form to display that same record.

Create an Event Procedure for the *On Click* event of the *OK* button and enter the code as follows (*Listing 4*):

Listing 4:

```
Private Sub cmdOK_Click()
    On Error Resume Next
    Dim rst As Object
    DoCmd.OpenForm "frmEmployees"
    Set rst = Forms![frmEmployees].RecordsetClone
    rst.FindFirst "[EmployeeID]=" & Me.lstSearch.Value
    Forms![frmEmployees].Bookmark = rst.Bookmark
    DoCmd.Close acForm, "frmSearch"
End Sub
```

How it Works:

The code first declares an Object variable that I have named *rst* to represent a recordset. When using an Object variable (as opposed to a data variable) you must use the *Set* keyword when assigning a value to it.

A code statement opens the Employees form. If the form is already open it simply gets the focus. The next statement defines the RecordsetClone of the Employees form as the *rst* recordset variable.

NOTE: A RecordsetClone is a copy of the form's own recordset that can be used to navigate or operate on a form's records independent of the form itself.

Using *FindFirst* the RecordsetClone is instructed to move to the first occurrence of the value of the ListBox in the [EmployeeID] field. This is the Primary Key field of the recordset so we know that there will be only one occurrence of that value. We also know that the specified value exists because the List Box list is generated from the same recordset as is used by the form.

When moving through a recordset Access uses a bookmark to keep track of its position. That bookmark now marks the position in the RecordsetClone of the item that the user selected from the List Box. The form's bookmark is currently on the record it happens to be displaying now. The next code statement synchronises the form's bookmark with that of the RecordsetClone which makes the form move to that same record, so that the form now displays the record that the user chose in the Search form.

The final code statement closes the Search form.

Step 3: Turn the Search Form into a Dialog Box

Although it functions correctly, our Search tool still looks like a regular Access form so, to finish the job, we need to make some settings for it to function as a dialog box.

With the form in Design View select the form itself by clicking the small rectangle in the upper left corner of the design window where the rulers meet (*Fig. 21*) so that a black dot appears, and go to the form's Property Sheet.

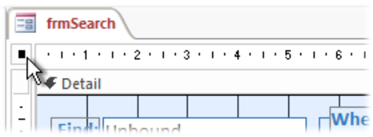


Fig. 21 Select the Form in Design View.

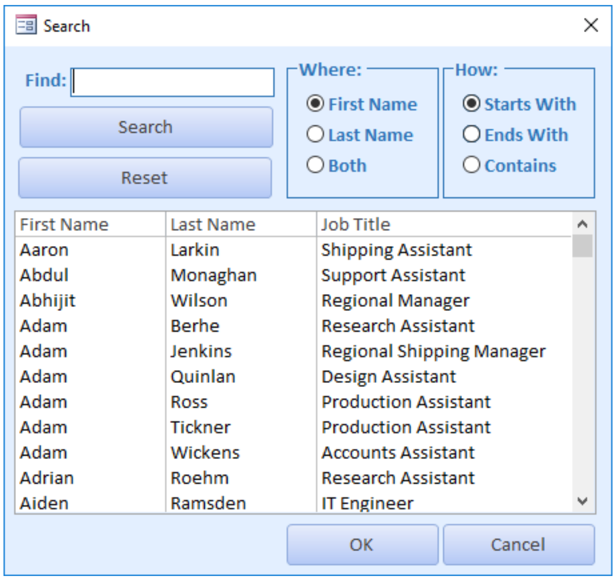
Make the following settings (*Table 2*):

Table 2: Form Properties

Property	Tab	Value
Pop Up	Other	Yes
Modal	Other	Yes (optional*)
Caption	Format	Search
Auto Center	Format	Yes
Border Style	Format	Dialog
Record Selectors	Format	No
Navigation Buttons	Format	No
Scroll Bars	Format	Neither

*NOTE: The Modal property, when set to Yes, prevents the user working outside the dialog box whilst it is open. I have marked this property as optional because, in this example, the tool closes after the user asks to view the selected record. You might choose to omit the statement that closes the Search tool so that the user can leave it open and search other records. In this case you might want to set the Modal property to No (the default setting).

When you view your dialog box in Form View it should now look something like this (*Fig. 22*):



Where: ● First Name ○ Last Name ○ Both

How: ● Starts With ○ Ends With ○ Contains

First Name	Last Name	Job Title
Aaron	Larkin	Shipping Assistant
Abdul	Monaghan	Support Assistant
Abhijit	Wilson	Regional Manager
Adam	Berhe	Research Assistant
Adam	Jenkins	Regional Shipping Manager
Adam	Quinlan	Design Assistant
Adam	Ross	Production Assistant
Adam	Tickner	Production Assistant
Adam	Wickens	Accounts Assistant
Adrian	Roehm	Research Assistant
Aiden	Ramsden	IT Engineer

Fig. 22 The completed Search dialog.

TIP: Having made these settings you might find that after viewing the Search tool in Form View the View button on the Access Home tab is disabled, preventing you from switching the dialog box into Design View. Instead, simply right-click on the dialog itself and choose Design View from the context menu.

Step 4: Add a Button to Open the Search Tool

The best way to open the Search tool is with a command button. I usually place one on the form that will display the relevant records and perhaps another on the database's Switchboard or Home screen. The process is the same wherever you choose to put it.

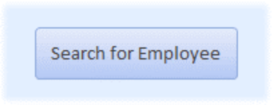


Fig. 23 Add a command button to open the Search tool.

Draw a command button on the form on which you want it to appear, give it a suitable name and caption (*Fig. 23*), and add the following code to its On Click event (*Listing 5*):

Listing 5:

```
Private Sub cmdSearchForEmployee_Click()
    On Error Resume Next
    DoCmd.OpenForm "frmSearch"
End Sub
```

Job done! At this point you have a fully functioning Search tool, but there are various things you can do to refine and widen the scope of the tool. Here are a few refinements to consider…

Additional Optional Refinements

A Non-Numeric Primary Key

I emphasised the need to include the Primary Key field in the Search tool's List Box Row Source because this is the most efficient way to search for a specific record. Every record will have a value in its Primary Key field. I usually use an *Autonumber* data type for the Primary Key field but that is not obligatory.

If your Primary Key field is not numeric you must modify the SQL statement to handle a text value. You will need to modify the code that powers the **OK** button (*Listing 4*) so that instead of:

```
    rst.FindFirst "[EmployeeID]=" & Me.lstSearch.Value
```

it reads:

```
    rst.FindFirst "[EmployeeID]=" & Chr(34) & Me.lstSearch.Value & Chr(34)
```

This simply places quote marks around the text value that comprises the text of your Primary Key.

The User Does Not Enter Anything in the Find Box

I included an error handler so that, in the event of the user clicking the Search button without entering anything in the Find text box, nothing will happen. You might want to add a message reminding the user that they need to make an entry in the Find text box first.

Modify the code behind the **Search** button (*Listing 3*) so that it includes the following (*Listing 6*):

Listing 6:

```
    strSearch = Me.txtSearch.Value
    If Len(strSearch) = 0 Then
        MsgBox "You must enter something in the Find box.", vbExclamation
        Me.txtSearch.SetFocus
        Exit Sub
    End If
```

The If Statement checks the length of the string in the *strSearch* variable which returns zero if the variable is empty indicating that the user did not enter anything. That being the case, the user is shown a suitable message after which their cursor is returned to the Find text box and the macro is cancelled.

The Selected Record Was Not Found

In this example the Search tool's list was created from the same table as that being searched so we can be certain that any of the items in the list will be found in the recordset we are searching. However, if the Search tool's list was created from a different recordset it's possible that the chosen record will not be found. The simple error handler prevents an error occurring if the specified value isn't found, and the bookmarks would simply synchronize at the end of the recordset. However, it would be better to let the user know what has happened so, if this might apply to your data, modify the code behind the **OK** button (*Listing 4*) as follows (*Listing 7*):

Listing 7:

```
    rst.FindFirst "[EmployeeID]=" & Me.lstSearch.Value
    If rst.EOF Then
        MsgBox "A matching record was not found.", vbInformation
        Exit Sub
    End If
```

If Access moves through the entire recordset without finding the specified value it finds itself at the end of the recordset, known in VBA as the *End of File* (EOF). The code uses an If Statement to check whether, having searched the recordset, it finds itself at the End of File position. If that is true then the user is shown a suitable message and the macro is cancelled.

Can I Use the Same Search Tool for Multiple Forms?

Yes! Your Search tool needs to know which form has opened it. To do this you need to create a Global Variable (i.e. one that does not reside inside a macro so that its value can be written and read from any macro).

In the Visual Basic Editor create a new Module and at the top enter the variable declaration:

```
    Public strCallingForm As String
```

I have named the variable *strCallingForm* but you can give it any name you like. The code behind each button that opens the Search tool on your various forms must now also place a value in this variable which is the name of the form that you wish to search, for example (*Listing 8*):

Listing 8:

```
Private Sub cmdSearchForEmployee_Click()
    On Error Resume Next
    DoCmd.OpenForm "frmSearch"
    strCallingForm = "frmEmployees"
End Sub
```

You also need to modify the code behind the Search tool's **OK** button so that it knows which form to search. Unfortunately you can't simply replace the name of our original form with the name of our variable. Instead, replace the original code with a case statement that adds a case for each possible calling form (*Listing 9*), for example:

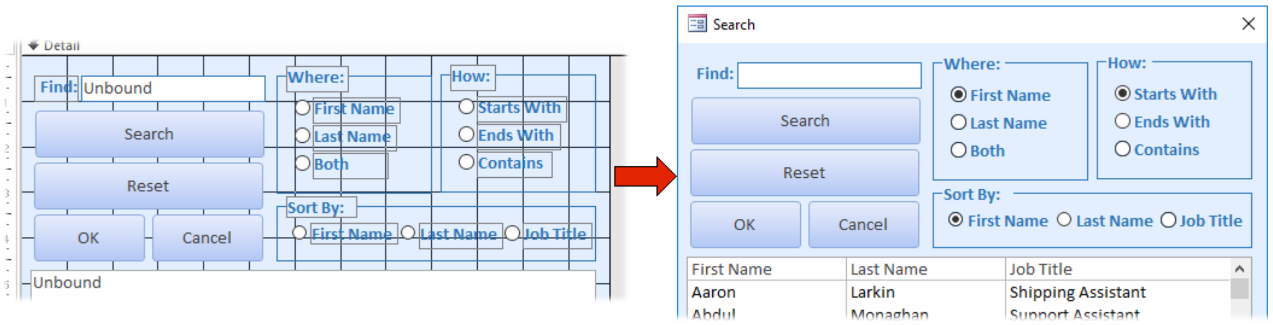
Listing 9:

```
Private Sub cmdOK_Click()
    On Error Resume Next
    Dim rst As Object
    Select Case strCallingForm
        Case "frmEmployees"
            DoCmd.OpenForm "frmEmployees"
            Set rst = Forms![frmEmployees].RecordsetClone
            rst.FindFirst "[EmployeeID]=" & Me.lstSearch.Value
            If rst.EOF Then
                MsgBox "A matching record was not found.", vbInformation
                Exit Sub
            End If
            Forms![frmEmployees].Bookmark = rst.Bookmark
        Case "frmEmployees2"
            DoCmd.OpenForm "frmEmployees2"
            Set rst = Forms![frmEmployees2].RecordsetClone
            rst.FindFirst "[EmployeeID]=" & Me.lstSearch.Value
            If rst.EOF Then
                MsgBox "A matching record was not found.", vbInformation
                Exit Sub
            End If
            Forms![frmEmployees2].Bookmark = rst.Bookmark
    End Select
    DoCmd.Close acForm, "frmSearch"
End Sub
```

Sort the List Box List

If you are dealing with a large number of records it might be easier for the user to find what they are looking for if you sort the list of results. I have included a standard ORDER BY clause in my example (*Listing 3*) but you could allow the user to choose the sort order. I am going to keep this simple by allowing the user to choose only one field.

To allow the user to choose which field to sort by the Search dialog needs another Option Group. I rearranged the objects on the dialog box to accommodate the new item (*Fig. 24*). I named the new Option Group *grpSort* and set its Default Value property to *1*.



First Name	Last Name	Job Title
Aaron	Larkin	Shipping Assistant
Abdul	Monaghan	Support Assistant

Fig. 24 Add a Sort option to the Search dialog.

Some modifications must be made to the original macro behind the Search button (*Listing 3*). First add a statement declaring a new string variable to hold the text that will form the fields of the ORDER BY clause of the SQL:

```
    Dim strSort As String
```

Create a new Case Statement to create appropriate SQL statements according to the user's choice from the *optSearch* option group (*Listing 10*). You will see that I have chosen to sort by multiple fields, starting with that chosen by the user. You can choose to do the same or simply sort by the one field that the user selected:

Listing 10:

```
Select Case Me.grpSort
  Case 1 'Firstname
    strSort = " tblEmployees.[FirstName],tblEmployees.[Lastname]"
  Case 2 'Lastname
    strSort = " tblEmployees.[Lastname],tblEmployees.[Firstname]"
  Case 3 'Jobtitle
    strSort = " tblEmployees.[Jobtitle],tblEmployees.[Firstname],tblEmployees.[LastName]"
End Select
```

Finally, modify the SQL statement that will be used as the List Box Row Source by replacing the existing ORDER BY fields with the *strSort* variable (*Listing 11*):

Listing 11:

```
strSQL = "SELECT tblEmployees.EmployeeID, tblEmployees.Firstname, " & _
         "tblEmployees.Lastname, tblEmployees.JobTitle " & _
         "FROM tblEmployees " & _
         "WHERE " & strSearch & " " & _
         "ORDER BY " & strSort & ";"
```

Writing Safe Code

When creating this Masterclass I decided to build two Search dialogs, one with and one without the above enhancements. I built the basic dialog box first then made a copy of it and added the enhancements to the copy. The original form was named *frmSearch* so I named the copy *frmSearch2*. But when I was testing the enhanced dialog I found that it didn't close when it was supposed to. I had forgotten to change the code that instructed the form, by name, to close. This reminded me why, when writing a code statement in which a form refers to itself, instead of writing:

```
    DoCmd.Close acForm, "frmSearch"
```

I usually write:

```
    DoCmd.Close acForm, Me.Name
```

This has the advantage that, if the name of the form is changed, the code will still work. I could say that, trying to keep things simple, I didn't do it. But I just forgot. Even an "expert" can make mistakes!

Download the Sample File

You can download a sample Access database containing the completed example of the Search tool built in this Masterclass. The database contains two copies of the Search tool: the basic dialog as described in the main part of the Masterclass, and an enhanced version containing the Optional Refinements described above.

Download the sample database file from:

http://www.fontstuff.com/access/files/AccessFormsMasterclass4.zip