Access Forms Masterclass 6
A Push-Button Filter for Your Access Forms

Published: 5 October 2018
Author: Martin Green
Screenshots: Access 2016, Windows 10
For Access Versions: 2007, 2010, 2013, 2016

I first came across this idea when learning Access 2 many years ago, so thanks to the author of whatever book I found it in (and apologies for not being to remember the name of the book or its author!). Then it was called a Rolodex, referring to the well-known card index gadget found on so many desks in those days. Not wanting to be sued by the Rolodex company for taking their excellent product's brand name in vain, I have chosen to call this tool a Push-Button Filter. The original tool was powered by Access macros. In those early Microsoft office days there was not the Visual Basic Editor we have today, and any code writing was in Access Basic. My version uses VBA and benefits from the power and flexibility this offers.

The idea is very simple. The tool takes the form of a console consisting of a collection of 26 *Toggle Buttons*, one for each letter of the alphabet (*Fig. 1*), enclosed in an *Option Group*.

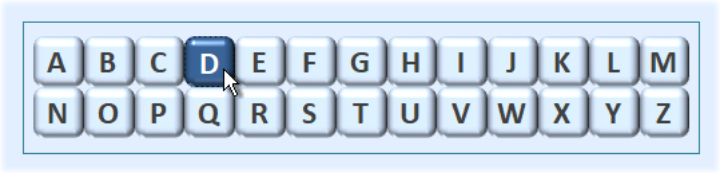

Fig. 1 The Push-Button Filter tool.

The user clicks a button, and something happens. Exactly what depends upon you. I'm offering a couple of suggestions. One idea is a Filter, so that the form's recordset is filtered to show only those records in which the text of a specific field starts with the chosen letter. Another idea makes use of a sorted recordset, in which the form displays the first record that starts with the chosen letter. There are lots more possibilities for a tool like this but once you understand the general principle of how it works you can probably come up with other ideas yourself. Regardless of how it is to be used, the method of building the Push-Button console is the same.

Step 1: Build the Push-Button Console

Depending upon how it will be used, the console can be placed on the form, or in its header or footer sections. Move your form into Design View and, in an empty space, use the *Option Group* tool (*Fig. 2*) to draw a rectangle large enough to enclose the required number of buttons (*Fig. 3*). Don't worry too much about the size, you can adjust it to fit as you go along.

Fig. 2 The Option Group tool.

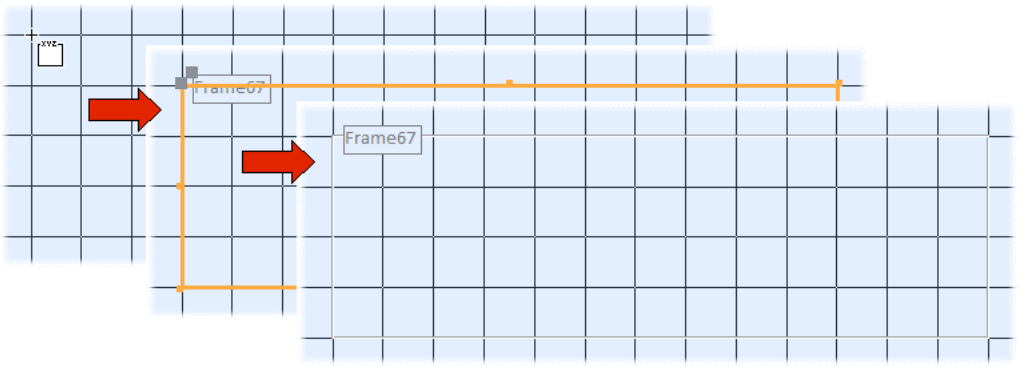

Fig. 3 Draw a rectangle on the form.

The Option Group needs a sensible name so enter one (I called mine *grpConsole*) in the **Name** textbox on the **Other** tab of its Property Sheet (open the Property Sheet with **[Alt]+[Enter]**).

Next, select the **Toggle Button** tool on the **Design** tab of the ribbon (*Fig. 4*) then click inside the rectangle you just created. As you hover over the Option Group frame its background turns black to indicate that the item you are about to drop onto it will form part of an option group (*Fig. 5*).

Fig. 4 The Toggle Button tool.

NOTE: Like *Option Buttons* and *Check Boxes*, *Toggle Buttons* have two states: selected and not-selected. When used alone on a form these states become *True* and *False* respectively, but when these controls are placed inside a *Frame* as part of an *Option Group* they each have a numerical value which, when selected, becomes the value of the *Option Group*. The numerical value is assigned automatically as you add controls to the group, incrementing from 1, or it can be assigned manually using the control's *Option Value* property.

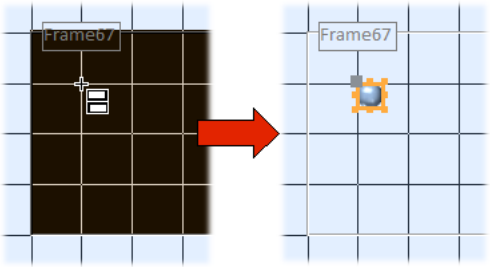

Fig. 5 Drop a Toggle Button onto the Frame.

Format the Toggle Button to suit the style of your own form. For this example, I used the following settings (*Table 1*) on the Toggle Button's Property Sheet (if necessary, open the Property Sheet by selecting the Toggle Button and pressing **[Alt]+[Enter]**).

Table 1: Properties of the first Toggle Button.

Property	Tab	Value
Width:	Format	1 cm
Height:	Format	1 cm
Back Color:	Format	Access Theme 3 (optional)
Caption:	Format	A
Font Size:	Format	18
Font Weight:	Format	Bold
Option Value:	Data	1

Having created the first Toggle Button (*Fig. 6 left*) you now need to add the remaining 25, one for each letter of the alphabet. You could add them one at a time, but I prefer to save time by copying and pasting. Simply click on the first Toggle Button to select it and press **[Ctrl]+C** then **[Ctrl]+V** to create an exact copy. Move the copy so that it sits alongside the first, then repeat until you have built the first row of buttons. Controls on a form will snap to the grid as you move or resize them so it's quite easy to align them neatly.

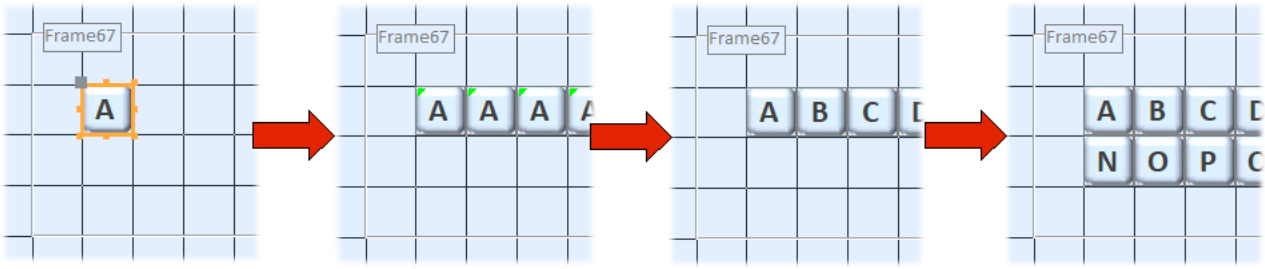

Fig. 6 Add the remaining Toggle Buttons to the Option Group.

Now you have a row of Toggle Buttons all labelled "A" so you need to change the *Caption* property of each one as appropriate. The copy/paste process also confers the same *Option Value* on each button so they currently all have the same value as the first one. Access has spotted this and has displayed a green marker on each to warn you that the value is duplicated (*Fig. 6 centre left*). Change the Option Value property of each button (B=2, C=3, D=4 and so on) to fix this (*Fig. 6 centre right*). As you do this the green warning markers disappear.

If, like me, you choose to arrange your buttons in more than one row, you can create the first row then copy it and paste the next row underneath. Change the captions and values as before until you have an entire alphabet of buttons (*Fig. 6 right*).

> TIP: When selecting multiple controls on a form, you can click on the first item then hold down **[Ctrl]** or **[Shift]** whilst you click on each of the others. If the controls to be selected are adjacent to each other, you can make a multiple selection by using the mouse to draw a box around the items to be selected.

If you wish you can add a caption to the Option Group's label. I preferred to delete the label (select the label then press **[Delete]**). I formatted the outline of the Option Group's frame to suit the design of my form by changing its *Special Effect* property to *Flat* and choosing a suitable colour for its *Border Color* property. I also adjusted the size of the frame to better suit the arrangement of the buttons (*Fig. 7*).

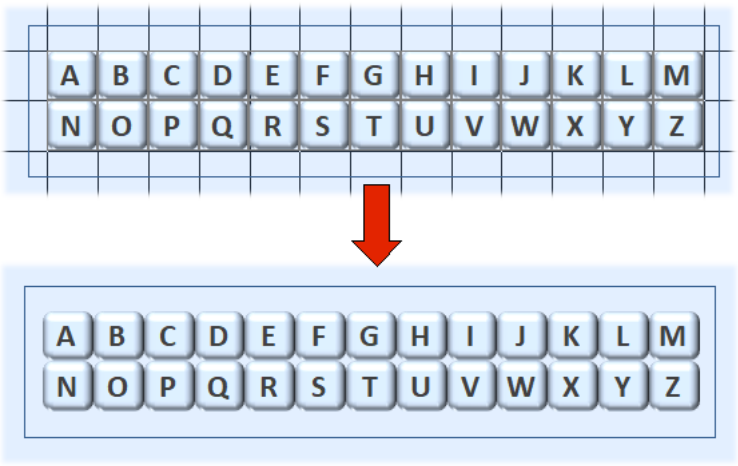

Fig. 7 The completed Option Group in Design View and Form View.

Once created, the button console can be moved to a suitable location. Start by making a multiple selection of the entire Option Group as described in the TIP above, then point at the outline of the selection so that the Mover cursor appears (*Fig. 8*) then simply drag the whole thing to its desired location.

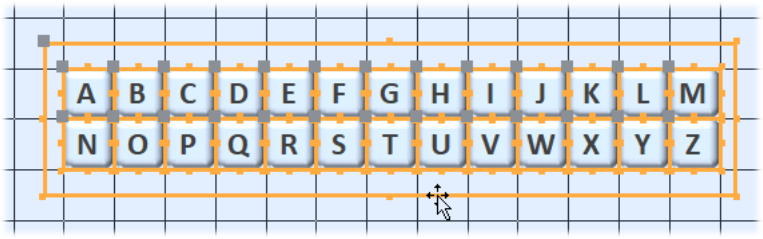

Fig. 8 Drag the Option group to its desired location.

For accurate placement use the **[Arrow]** keys to move the selected items up, down, left or right. Hold down the **[Control]** key whilst doing so for fine-tuning.

> TIP: I have found the most useful command when building forms is **Undo**! It is so easy to accidentally move, change or delete an object. Don't panic! Just press **[Ctrl]+Z** until you have undone whatever it is that you didn't mean to do.

The Push-Button console is now complete. When in form view you will see that only one button can be selected at a time. If you select another button the previously selected button will be deselected.

The next task is to write the VBA code that will turn it into a useful tool.

Step 2: Write the VBA Code

I am suggesting two different uses for this Push-Button console:

- **Filter by Letter** When a letter is chosen the form's recordset is filtered by a particular field so that only those records with entries starting with that letter are shown.
- **GoTo by Letter** This requires the recordset to be sorted alphabetically (A-Z) by the field in question. When the user chooses a letter, the form moves to the first record that starts with that letter.

A Filter by Letter Tool

1. Prepare the Form

To best appreciate this tool in action I have created a form that displays its records as *Continuous Forms*. I designed the form with the fields arranged in a single row so that, in form view the form has the appearance of a datasheet. So, why not use Datasheet View? Datasheet View does exactly what it says and presents the data in the form of a datasheet and does not allow the addition of a header or footer, so there is nowhere to place the Option Group. I can achieve the same effect with a little work on a Continuous Forms form and still have somewhere to locate the console (*Fig. 9, Fig. 10*).

Note that I have also added a "Show All" button to allow the user to cancel the filter if they wish.

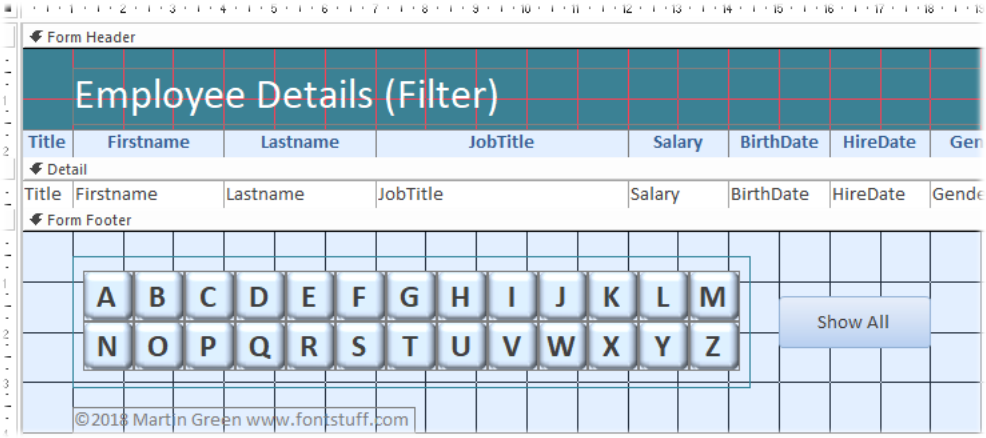

Fig. 9 The Continuous Forms form in design view.

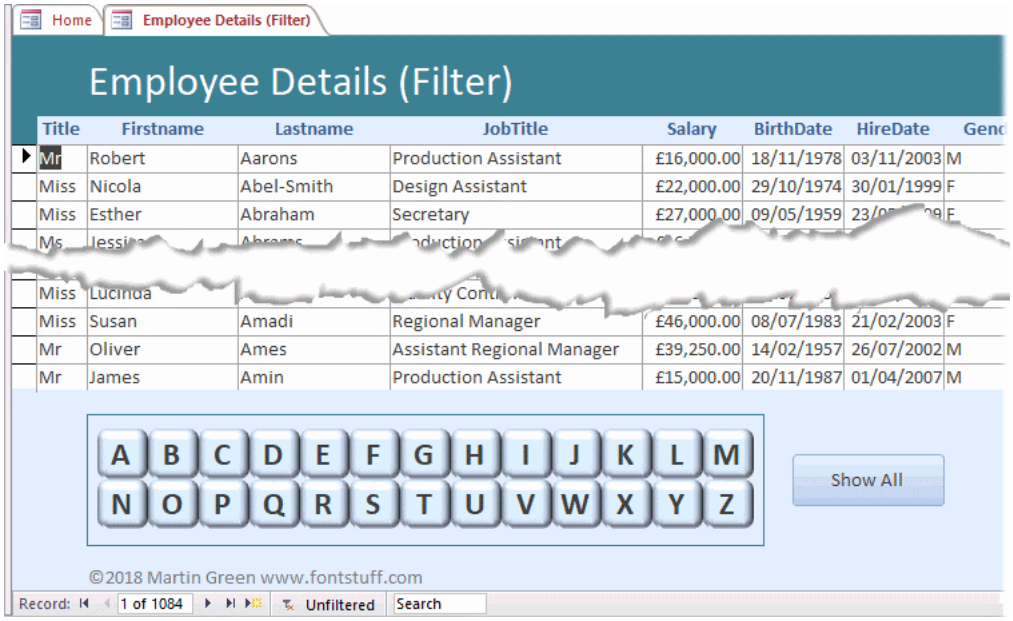

Fig. 10 The Continuous Forms form in form view.

2. Enter the Code

The macro that applies the filter must run when one of the console buttons is pressed. Pressing a button results in the value of the Option Group changing, adopting the *Option Value* of the selected button. The code is therefore attached to the *AfterUpdate* event of the Option Group control.

Select the Option Group by clicking on its frame (not on one of the buttons) and click in the **AfterUpdate** text box on the **Event** tab of its Property Sheet. Click the **Build** button (**[...]**) and choose **Code Builder** in the *Choose Builder* dialog then click **OK** to open to the Visual Basic Editor where an empty *AfterUpdate* macro has been created. Add the following code (*Listing 1*) remembering to change the name of the Option Group if you used something different:

Listing 1:

```
Private Sub grpConsole_AfterUpdate()
    On Error Resume Next
    Dim strLetter As String
    strLetter = Chr(64 + Me.grpConsole.Value)
    Me.OrderBy = "[Lastname], [Firstname]"
    Me.OrderByOn = True
    Me.Filter = "[Lastname] Like " & Chr(34) & strLetter & Chr(42) & Chr(34)
    Me.FilterOn = True
End Sub
```

3. How It Works

I can't think of anything that might go wrong here but to be safe the code starts with a basic error handler.

The code needs to convert the number that is the value of the Option Group into a letter. I am going to store that letter in a string variable called *strLetter* so the next statement declares the variable.

The code needs to change the numerical value of the Option Group (e.g. A=1, B=2 etc.) into the letter it represents, and makes use of the *Chr()* function to do this. The ASCII value of the letter "A" is 65. These ASCII values increment by one through the alphabet, so that "Z" is character 90.

> NOTE: Upper-case and lower-case letters have different ASCII values. Lower-case "a" is character 97, through to "z" being character 122. Since the filter is not case sensitive it doesn't matter which we use. I have chosen upper-case.

I could have applied the actual ASCII values to the buttons, but it is a simple matter to convert them. All that is needed is to add 64 to the value of the Option Group to get the ASCII value of the letter corresponding to the button that was pressed. The resulting number is then put into the *Chr()* function which is then passed to the *strLetter* variable (e.g. "A" = *Chr(64+1)*, "B" = *Chr(64+2)* etc.).

The filter doesn't require the recordset to be sorted but it makes sense to sort the results of the filter, so the next two statements apply a sort order and switch it on.

Finally, two statements construct the filter, apply it and switch it on. To build the filter I have made use of the *Chr()* function again to represent various characters (for example *Chr(34)* is the quote mark (") and *Chr(42)* the asterisk (*****))

> TIP: The *Chr()* function is very useful when constructing text strings in VBA, especially when working with SQL where quote marks, asterisks etc. are involved. To find out the ASCII value of any character (i.e. the number you give the *Chr()* function to generate a character), open the Visual Basic Editor's Immediate Window (keys: **[Ctrl]+G**) and type, for example, **?Asc("A")** and press **[Enter]**. The *Asc()* function will return the ASCII value of the character you gave it.

4. Suggested Improvements

You could add a couple of simple improvements to this macro. If the user chooses a letter for which there are no records the form will simply display a single empty record. Rather than leave it at that, I prefer to let the user know that this is because no matching records were found. I use an If Statement and a Message Box to achieve this (*Listing 2*).

Listing 2:

```
If Me.Recordset.RecordCount = 0 Then
    MsgBox "No records starting with " & strLetter & " were found.", vbInformation
End If
```

The *If Statement* checks the *RecordCount* property of the form's *Recordset* property which holds the number of records currently being displayed. When executed after the filter has been applied it will give the number of records that remain *after* the filter was applied. If the result is zero, then a simple message is displayed (*Fig. 11*).

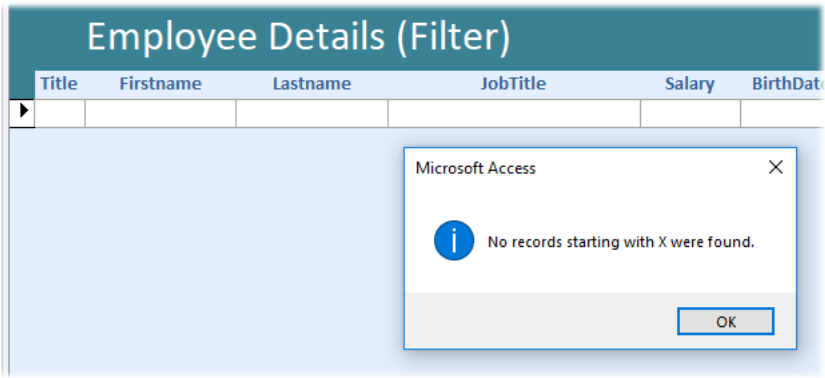

Fig. 11 An optional message shows that nothing was found.

You might notice that, when a button is pressed, the screen flickers as the filter is applied and the picture redrawn. Whilst this is a minor inconvenience user might find it distracting. It is easily cured by hiding screen updates until the process has finished. Do this by turning *Echo* off then on again. The completed macro including the message and *Echo* commands is shown here (*Listing 3*):

Listing 3:

```
Private Sub grpConsole_AfterUpdate()
    On Error Resume Next
    Dim strLetter As String
    strLetter = Chr(64 + Me.grpConsole.Value)
    DoCmd.Echo False
    Me.OrderBy = "[Lastname], [Firstname]"
    Me.OrderByOn = True
    Me.Filter = "[Lastname] Like " & Chr(34) & strLetter & Chr(42) & Chr(34)
    Me.FilterOn = True
    DoCmd.Echo True
    If Me.Recordset.RecordCount = 0 Then
        MsgBox "No records starting with " & strLetter & " were found.", vbInformation
    End If
End Sub
```

NOTE: When using *DoCmd.Echo False* you must remember to switch it on again before the end of the macro. It is also vital that your code is protected by an error handler. A simple error handler like that used here is fine for the simplest of macros where nothing serious could happen. When using a full error handler, you must include *DoCmd.Echo True* in the Exit Routine to make sure that, in the event of an error, screen updating is restored. Failure to do this could leave your user with a blank screen.

5. Add a Show All Button

Unless you set a default value for the Option Group it will have a value of zero when the form opens and the form's recordset will be unfiltered. If the user wants to remove the filter they can do so by clicking the "Filtered" button next to the navigation buttons at the bottom of the screen, or the *Toggle Filter* button on the Home tab of the ribbon. However, doing this does not affect the Option Group and the last-chosen button will remain selected. This might give the impression that the recordset was filtered when it wasn't. You could attempt to synchronise the button console with the recordset but, wanting to keep things simple, I have chosen to add a "Show All" button.

This is simply a command button (I named mine *cmdShowAll*) with a command to switch off and clear the form's filter applied to its *OnClick* event (*Listing 4*).

Listing 4:

```
Private Sub cmdShowAll_Click()
    On Error Resume Next
    Me.Filter = ""
    Me.FilterOn = False
    Me.grpConsole.Value = 0
End Sub
```

It also includes a command setting the value of the Option Group to zero so that no buttons appear selected. Unlike the built-in commands, it doesn't toggle the filter on and off, it simply makes sure it's off.

A GoTo by Letter Tool

This tool is intended for use on forms displaying their records in *Single Form* view. It also requires that the recordset be sorted alphabetically (A-Z) but the code will take care of that. The console can be placed anywhere on the form, or in its header or footer sections. In this example I have placed it in the form's footer (*Fig. 12*).

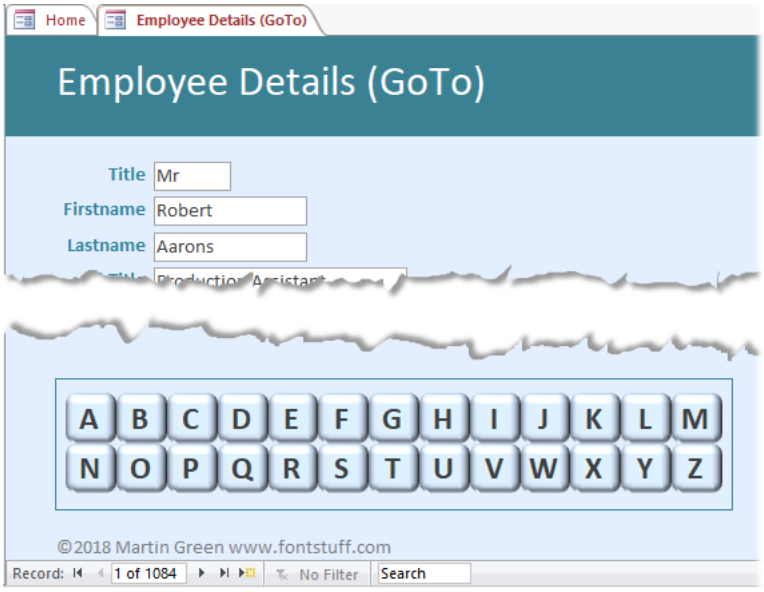

Fig. 12 The console on a Single Form form.

When the user clicks a button on the console the form will move to the first record in the recordset whose entry in the specified field (in this case *Lastname*) starts with the chosen letter. The user can then move through the records in the usual way (they will have been sorted alphabetically by the macro) or select another letter.

1. Enter the Code

As with the *Filter by Letter* tool the code runs on the *AfterUpdate* event of the Option Group (*Listing 5*):

Listing 5:

```
Private Sub grpConsole_AfterUpdate()
    On Error Resume Next
    Dim strLetter As String
    Dim rst As Object
    strLetter = Chr(64 + Me.grpConsole.Value)
    Me.OrderBy = "[Lastname] ASC"
    Me.OrderByOn = True
    Set rst = Me.RecordsetClone
    rst.FindFirst "[Lastname] Like " & Chr(34) & strLetter & Chr(42) & Chr(34)
    If rst.NoMatch Then
        MsgBox "No records starting with " & strLetter & " were found.", vbInformation
    Else
        Me.Bookmark = rst.Bookmark
    End If
    Set rst = Nothing
End Sub
```

2. How It Works

As in the previous example the macro notes which button has been pressed and in the same way stores it in the *strLetter* variable. It then sorts the form's recordset by the *Lastname* field in ascending (A-Z) order.

The code makes use of the form's *RecordsetClone* property which it stores in an object variable named *rst*.

> NOTE: The Recordset Clone is an exact copy of the form's recordset that can be manipulated in memory without interfering with the recordset that the form is displaying. Having carried out any necessary work on the clone it can be discarded. When assigning a value to an object variable, such as the Recordset Clone, you must use the *Set* keyword.

The macro uses the *FindFirst* method to travel through the recordset until it finds a record that satisfies the specified condition, in this case a record in which the *Lastname* field starts with the chosen letter. The recordset must be in ascending sort order for this to work.

If the end of the recordset is reached without a matching record being found the recordset's *NoMatch* property is set to *True*. An If Statement checks to see whether this is the case and if so displays a message stating that nothing was found. If, however, a suitable record is found the *FindFirst* method stops on that record and its position in the recordset is bookmarked.

The position of the form's bookmark is then synchronised with that of the Recordset Clone's bookmark, making the form display the found record. Finally, the *rst* variable is set to *Nothing* to clear it from memory.

3. A Suggested Improvement

Having found their chosen records, the user might move to other records in the recordset. As they do so the console will continue to display the letter the user last chose, even though they might be viewing a record with a *Lastname* starting with a different letter.

The solution is to synchronise the console with the record that is currently being displayed. The following code runs on the form's *Current* event (*Listing 6*). This event happens when the form opens and each time a different record is displayed.

Listing 6:

```
Private Sub Form_Current()
    On Error Resume Next
    If IsNull(Me.Lastname.Value) Then
        Me.grpConsole.Value = 0
    Else
        Me.grpConsole.Value = Asc(UCase(Left(Me.Lastname.Value, 1))) - 64
    End If
End Sub
```

The code uses an If Statement to check if the *Lastname* field contains a value. If not (i.e. its value is *Null*) the value of the Option Group is set to zero so that no buttons are pressed. If there is a value in the *Lastname* field the code determines the ASCII value of the first (leftmost) letter then subtracts 64 to arrive at a number it can assign to the Option Group so that the correct button appears selected.

> NOTE: when searching a recordset, text is normally not case sensitive, but when determining the ASCII value of a character upper-case and lower-case letters have different values. To prevent confusion, I have used the *UCase* function to ensure that, whatever case the *Lastname* data is written in, the ASCII value of the upper-case version of the first letter is calculated.

Download Example Database

You can download a sample Access database containing completed examples of the form shown in this Masterclass. The database contains two copies of the form: one equipped with the *Filter by Letter* tool and the other with the *GoTo by Letter* tool.

Download the sample database file from:

http://www.fontstuff.com/access/afmc06.htm