Access Forms Masterclass 1
Custom Navigation Buttons

Published: 1 August 2013
Author: Martin Green
Screenshots: Access 2010, Windows 7
For Access Versions: 2007, 2010, 2013

Why Add Custom Navigation Buttons?

Unless you specify otherwise, each Access form comes with its own set of built-in navigation buttons, located in the lower left corner of the form (*Fig. 1*) so why bother creating your own custom navigation buttons?

Fig. 1 A form's built-in Navigation Buttons.

When building applications for other people to use you soon learn that you should take nothing for granted and, most importantly, you should never assume any particular level of knowledge or expertise on the part of the user. Of course, you and I know what those little arrows at the bottom of a form are for and how to use them but what about the people who are going to use the database? Adding some clearly marked buttons might help them navigate through their records.

I have found that even knowledgeable users find custom buttons useful because on modern high resolution computer screens the built-in navigation buttons appear very small and can be difficult to see and operate.

I usually add my own custom navigation buttons to a form (*Fig. 2*) simply because it makes things easier and more convenient for the user. Whether or not you choose to keep the built-in navigation buttons is entirely up to you.

Fig. 2 Custom Navigation Buttons.

In this Masterclass you will learn how to add a set of custom navigation buttons to a form and write the VBA code to activate them. You can optionally remove the built-in navigation buttons and add refinements such as Tool Tips and code to disable the buttons when they are not needed.

Step 1: Add a Footer to the Form

First, you need to decide where to put your navigation buttons. I like to put them in the form's Footer section. The footer is always visible in the form's window so, if the form is too big for the window, the user won't have to scroll to find the navigation buttons.

When you create an Access form it doesn't have a footer by default so you have to add one. In the form's **Design** view Right-Click in the background of the form (the *Detail* area) and choose **Form Header/Footer** from the context menu (*Fig. 3*).

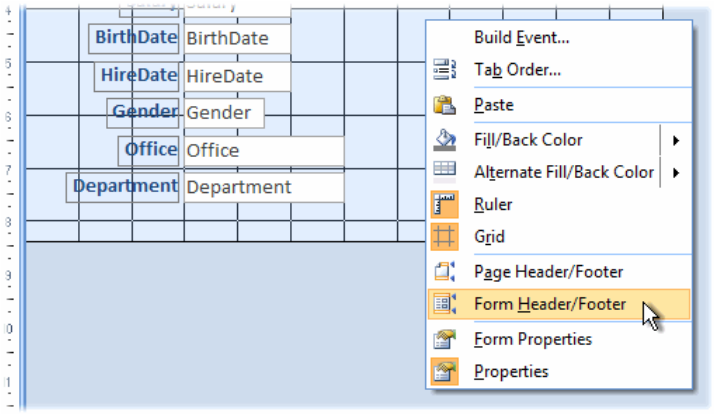

Fig. 3 Add a Footer to the form.

A Form Footer section appears at the bottom of the form (*Fig. 4*). Notice that a Form Header section is also created at the top of the form.

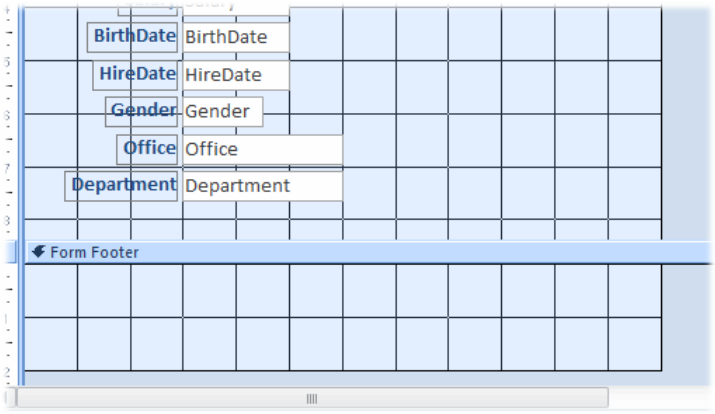

Fig. 4 A Footer section is added to the form.

If you don't need a header you can remove it by reducing its height to zero. To do this point at the upper edge of the bar labelled **Detail** so that the mouse pointer changes to a cross with a double headed vertical arrow (*Fig. 5*) then drag the bar up as far as it will go. This hides the Header section from view (*Fig. 6*).

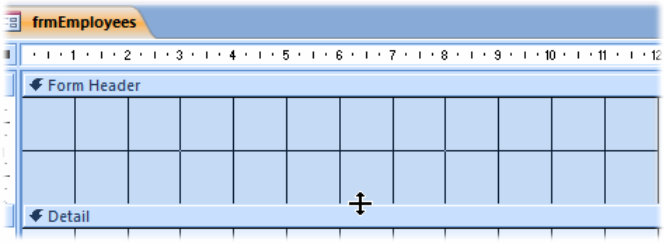

Fig. 5 Adjust the height of the Header section.

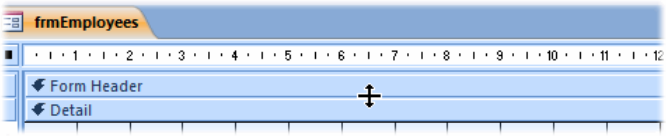

Fig. 6 Hiding the Header by reducing its height to zero.

If at any time you want to restore the Header just reverse the process. An alternative way to do this is to specify the **Height** property of the Header in the Property Sheet. Click on the bar labelled *Form Header* then go to the Property Sheet (click the **Property Sheet** button on the **Design** tab of the Ribbon or press **[Alt]+[Enter]** to open the Property Sheet if it is not already visible). Enter the required size in the **Height** property on the **Format** tab of the Property Sheet. Enter a zero to hide the Header or a number to specify the required size.

Step 2: Draw the Command Buttons

Having decided where to place them, the next step is to draw the required number of buttons on the form. Click the **Button** tool on the **Design** tab of the Ribbon (*Fig. 7*) then click on the form Footer approximately where you want the button to appear. I normally work with the Control Wizards tool switched off because I prefer to write my own code rather than let Access do it. If you have the Control Wizards tool switched on the Command Button Wizard will appear when you click on the footer. You could use the wizard to achieve the task in hand but since the point of this exercise is to teach you how to do these things yourself you can dismiss the wizard by clicking its Cancel button.

*NOTE: If you want to turn off Control Wizards permanently (you can turn them on again at any time) expand the **Controls** group on the **Design** tab of the Ribbon and de-select the **Use Control Wizards** option.*

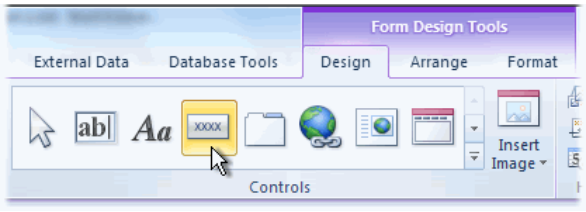

Fig. 7 Use the Button tool to add a Command Button.

When you clicked on the footer Access created a *Command Button* and gave it a default name and caption. In this example the button was given the name and caption *Command16* (*Fig. 8*). Whenever a control (an object on a form is called a *Control*) is added to a form Access automatically names it and, if appropriate, captions or labels it with a name and sequential number. You will give the button a sensible name and a meaningful caption later. First, you need to resize the button and position it accurately.

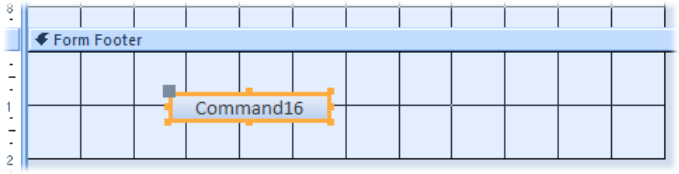

Fig. 8 The Button tool creates a new Command Button.

See that the when the Command Button control is selected it has a highlighted border with dots at the corners and mid-way along its sides. These are handles for resizing or moving the control using the mouse. Drag the small dots to change the size of the control. Drag the large dot at the upper left corner of the control to move it. I prefer to use either the keyboard or the Property Sheet to accurately size and position controls.

To use the keyboard to move a control, first click on the control to select it then press the **[Left Arrow]**, **[Right Arrow]**, **[Up Arrow]** or **[Down Arrow]** keys. Each button press moves the control one grid unit. For finer movements hold down the **[Control]** key whilst pressing the arrow key. This moves the control one quarter of a grid unit for each key press and is useful for accurate placement of items on the form.

To use the keyboard to change the size of a control, hold down the **[Shift]** key whilst pressing the arrow key. **[Shift]+[Left Arrow]** reduces the width of the control whilst **[Shift]+[Right Arrow]** increases the width. Similarly, **[Shift]+[Up Arrow]** reduces the height of a control whilst **[Shift]+[Down Arrow]** increases the height. Hold down the **[Control]** key as well to achieve fine changes.

Alternatively you can do it all using the Property Sheet. That's my preferred method. Select the Command Button control and go to the **Format** tab of the Property Sheet. Because I'm English I'm going to use centimetres but if you use inches just enter the nearest equivalent measurement that suits you. With the command button selected, go to the Property Sheet and enter the following property values (*Table 1*):

Table 1: Command Button "Back" Properties

Property	Tab	Value
Width:	Format	3 cm
Height:	Format	1 cm
Top:	Format	0.5 cm
Left:	Format	1 cm

There is no need to add "cm" or "in" as Access will do this automatically. You might also notice that, particularly if using metric measurements, Access changes some of the dimensions. For example 0.5 gets changes to 0.501. Don't worry about this, it's just Access converting measurements behind the scenes.

Give the command button a sensible caption by entering < *Back* in the **Caption** property, then switch to the **Other** tab of the Property Sheet and change the **Name** property to *cmdBack*. You now have a command button of the required size, in the desired position, with a meaningful name and caption (*Fig. 9*).

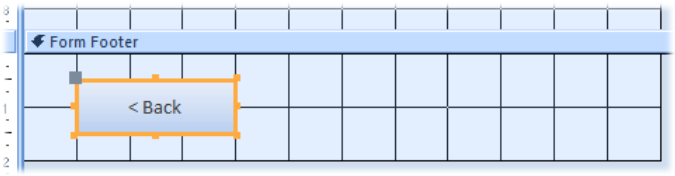

Fig. 9 The command button is positioned, resized and captioned.

You need two more buttons, one for *Next* and another for *New*. Since these need to be the same size as the one you have already made the simplest way to create them is to make copies of it. It doesn't matter how you do this. I like to use keyboard shortcuts, so with the first command button selected I press **[Control]+[C]** to copy it then **[Control]+[V]** twice to make two copies. Alternatively you could right-click the button and choose **Copy** then **Paste** from the context menu, or use the **Copy** and **Paste** commands on the **Home** tab of the Ribbon.

The new command buttons have been assigned new default names by Access but, like the original, they are all captioned < *Back*. The next task is to change their names and captions and place them in their correct locations. This can all be done from the Property Sheet.

Select the first of the new buttons and go to the Property Sheet and enter the following property values (*Table 2*):

Table 2: Command Button "Next" Properties

Property	Tab	Value
Name:	Other	cmdNext
Caption:	Format	Next >
Top:	Format	0.5 cm
Left:	Format	4 cm

Select the second of the new buttons and go to the **Property Sheet** and enter the following property values (*Table 3*):

Table 3: Command Button "New" Properties

Property	Tab	Value
Name:	Other	cmdNew
Caption:	Format	New
Top:	Format	0.5 cm
Left:	Format	7 cm

Depending on which method you used to create the new buttons, you might find that the Footer section increased in height to accommodate them. If necessary adjust the height of the Footer

section by pointing at its bottom edge so that the mouse pointer changes to a cross with a double headed vertical arrow (*Fig. 10*) then drag the border upwards to change the Footer to the desired size.

If you prefer, you can select the Footer by clicking the bar labelled *Form Footer* before entering the desired value in the **Height** property on the Property Sheet.

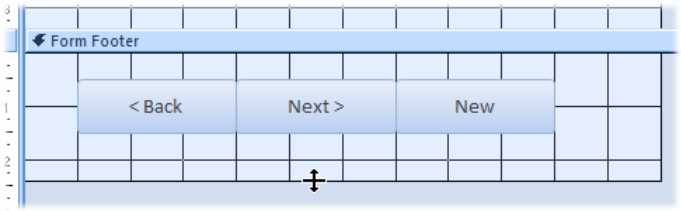

Fig. 10 Resizing the Footer section.

Before proceeding, switch the form into *Form View* and take a look at your new buttons (*Fig. 11*). If you want to change their size or position you can return to *Design View* to make any necessary adjustments.

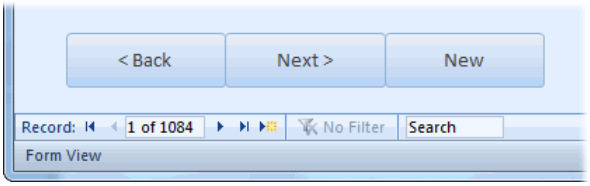

Fig. 11 The new Navigation Buttons in Form View.

Step 3: Additional Design Refinements

Add Tool Tips

Users might find it helpful to have a little extra information about what the buttons do. You can easily provide an additional hint by creating a *Control Tip*. This is a Tool Tip that appears when the user points their mouse at a control (*Fig. 12*). Enter some text in the **ControlTip Text** property (located on the **Other** tab of the Property Sheet) of each button. For example you might enter *Go To Previous Record* for the *cmdBack* button, *Go To Next Record* for the *cmdNext* button and *Create New Record* for the *cmdNew* button.

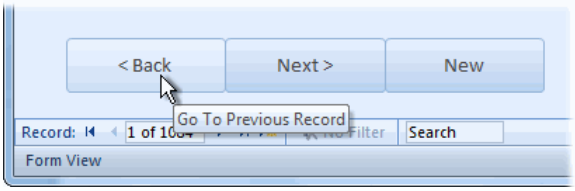

Fig. 12 A tool Tip helps users understand a control's function.

Check the Tab Order

A form's *Tab Order* is often overlooked by form designers. Many users navigate around a form by using their **[Tab]** key. This should take the user from control to control in a logical sequence. A control's position in the Tab Order is defined by the order in which that particular control was created on the form so, unless you have created each control in the precise order in which the user would be expected to visit them, the Tab Order will be incorrect. This can be very frustrating for someone who is used to using the **[Tab]** key to navigate around a form. It is very simple to check the Tab Order and, if necessary, correct it.

Each part of the form has its own Tab Order. To check the Tab Order of your new buttons, right-click on the background of the Footer section and choose **Tab Order** from the context menu (Fig. *13*). Alternatively, select the Footer by clicking on the bar marked *Form Footer* and click the **Tab Order** button on the **Design** tab of the Ribbon.

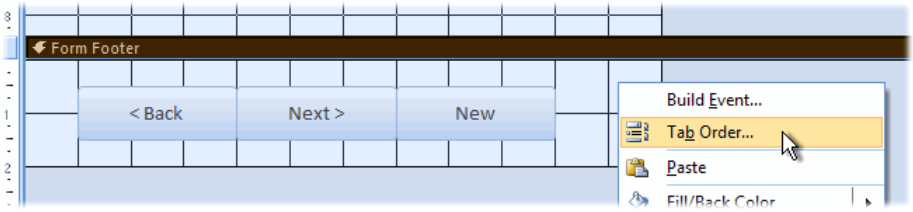

Fig. 13 Open the Tab Order dialog.

The Tab Order dialog (*Fig. 14*) shows the current order of controls in the selected Tab Order. To change the order, select a list item by clicking the grey button next to its name. Release the mouse then drag the selected item up or down the list to create the desired order. You can select multiple items and move several at once if required.

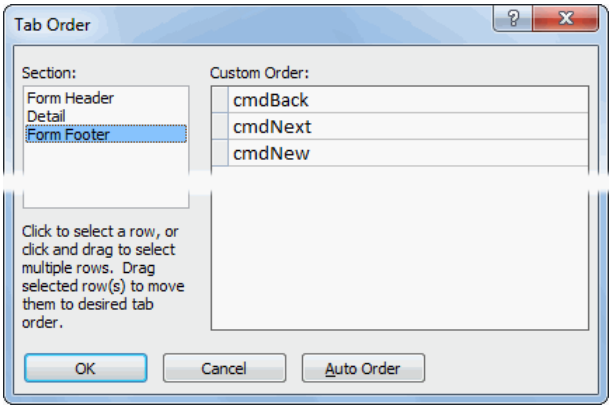

Fig. 14 The Tab Order dialog.

You can use the **Auto Order** command to have Access automatically create a tab order. If there is more than one column of controls on a form the automatic tab order will visit items left-to-right before moving down.

You should always check the Tab Order after making changes to a form. A simple change such as converting a Text Box control to a Combo Box will change its position within the Tab Order, moving it to the end of the order even though its position on the form has not changed.

Add First and Last Record Buttons

I haven't included *First* and *Last* record buttons because I seldom use them myself, but it is a simple matter to add them if you wish (*Fig. 15*).

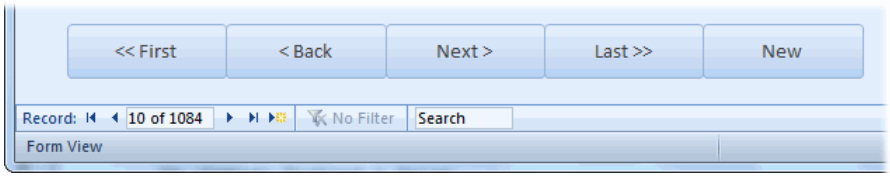

Fig. 15 First and Last buttons can be added.

Proceed exactly as for the other buttons, supplying appropriate names, captions and, if you are using them, tool tips for the command buttons. Remember to include them in the correct Tab Order.

Remove the Built-In Navigation Buttons

Now that you have your own custom navigation buttons you might like to remove the built-in ones. It's a simple process. In *Form Design* view, open the Property Sheet and select **Form** from the drop-down list at the top. Then, on the **Format** tab change the **Navigation Buttons** property to *No*. This causes the navigation bar to be hidden on that particular form (*Fig. 16*).

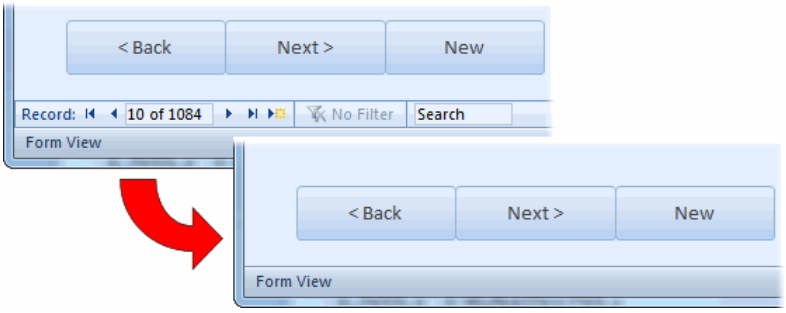

Fig. 16 Hiding the form's built-in Navigation Buttons.

Bear in mind that removing the built-in navigation buttons also removes the Record Count and Search boxes, so you might want to keep them in addition to your own. There are alternative methods for searching and in the next Masterclass I show you how to display your own custom Record Counter.

Step 4: Coding the Navigation Buttons

The buttons don't work yet. To activate them you need to write some VBA code. The code for each button will take the form of an *Event Procedure*, that is a procedure (commonly called a *Sub* or *VBA Macro*) that will run by itself when a particular event happens. In the case of our buttons that will be the *Click* event, which fires when the user clicks a button or presses their **[Enter]** key when a button is selected.

Coding the Back Button

In *Form Design* view select the cmdBack button and go to the **Event** tab of the Property Sheet. Click in the text box next to **On Click** then click the **Build** button (**[...]**) to open the **Choose Builder** dialog then select **Code Builder** (*Fig. 17*) and click **OK**.

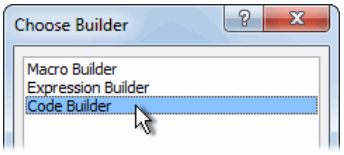

Fig. 17 The Choose Builder dialog.

This takes you into the *Visual Basic Editor* where Access has created an empty event procedure for the *cmdBack_Click* event, ready for you to add your code.

When writing VBA code you should always consider what might go wrong and cause an error when the code runs. If, for example, the user clicks the Back button when they are already on the first record this will cause the code to "crash". So, the first thing you need to do is add a simple error handler telling Access to ignore an error if one should occur when the user clicks the button.

(*NOTE: Error handlers usually need to be more sophisticated than this but in this case it will be safe to simply ignore the error.*)

Enter the code as follows:

1. Place the cursor between the lines beginning *Private Sub…* and *End Sub* and press **[Tab]** to indent your typing by one tab-space. Type **On Error Resume Next** then press [Enter] to create a new line.
2. Type **DoCmd** followed by a dot (**.**). When you type the dot Access shows you a list of all the available relevant commands.
3. Scroll down the list and choose **GoToRecord** (*HINT: Start typing the text you need and the list will automatically scroll to it.*). Either double-click the list item or select it and press **[Tab]** to add it to your code.
4. Type a **[Space]**. Access shows a list of possible objects, but since we are referring to the current form we don't need to specify this so skip this piece of information by typing a comma (**,**).
5. Access now wants to know the name of the object but, again, since we are referring to the current form we don't need to specify this, so skip this piece of information by typing another comma (**,**).
6. Finally we get a list allowing us to specify which record we want to go to. Double-click **acPrevious** to choose it.

Your finished code should look like this (*Listing 1*).

Listing 1:

```
Private Sub cmdBack_Click()
    On Error Resume Next
    DoCmd.GoToRecord , , acPrevious
End Sub
```

Before testing your code, first check your typing then open the Visual Basic Editor's **Debug** menu and choose **Compile...** (the name of your database is shown). Compiling the code checks for any errors you might have missed. If everything is OK (the Visual Basic Editor will tell you if it finds a problem) then click the **Save** button. This is an important step because, if you test code without saving your database first and an error causes Access to crash, you might lose some of your work.

Return to your database, put the form into *Form View* and test the *Back* button. Clicking it should move you to the previous record unless you are on the first record, in which case nothing should happen.

Coding the Next Button

Proceed exactly as described for the *Back* button. Create a *Click* event procedure for the *cmdNext* button and enter the code, this time specifying **acNext**. Your code should look like this (*Listing 2*):

Listing 2:

```
Private Sub cmdNext_Click()
    On Error Resume Next
    DoCmd.GoToRecord , , acNext
End Sub
```

Compile, save and test the code as before.

Coding the New Button

Proceed exactly as described for the *Back* button. Create a *Click* event procedure for the *cmdNew* button and enter the code, this time specifying **acNewRec**. Your code should look like this (*Listing 3*):

Listing 3:

```
Private Sub cmdNew_Click()
    On Error Resume Next
    DoCmd.GoToRecord , , acNewRec
End Sub
```

Compile, save and test the code as before.

Step 5: Additional Code Refinements

I suggested that you added a simple error handler in case the user pressed a button and gave Access a command that it could not complete. The error handler dealt with any possible error by ignoring it. But it is always good practice in programming to try to prevent errors happening in the first place. Often errors can't be anticipated so error handlers are always a good idea, but in this case we can prevent errors from happening by disabling those buttons whose function is not relevant at the time.

We can use the form's *CurrentRecord* property, which returns the index number of the record within the current recordset, to know if we are on the first record (the *CurrentRecord* property will have a value of *1*).

The *RecordCount* property of the form's recordset can be used to find out how many records there are and can be compared with the *CurrentRecord* to determine if we are on the last record or a new record (if the value of the *CurrentRecord* property is equal to the value of the *RecordCount* property then we are on the last record).

The form's *NewRecord* property can be used to determine if we are on a new record. It returns *True* for a new record and *False* for an existing record.

With this information we can change the *Enabled* property of the buttons to *True* or *False* to enable or disable the appropriate buttons so there is no chance of an error occurring.

To achieve this we make use of the form's *Current* event. This event fires whenever a new record is displayed. This happens when the form opens and when the user moves from record to record, or asks for a new record. To create the event procedure select the form itself by choosing **Form** from the drop-down list at the top of the Property Sheet. Select *OnCurrent* and proceed as before, entering code so that the finished procedure looks like this (*Listing 4*):

Listing 4:

```
Private Sub Form_Current()
    On Error Resume Next
    If Me.CurrentRecord = 1 Then
        Me.cmdBack.Enabled = False
    Else
        Me.cmdBack.Enabled = True
    End If
    If Me.CurrentRecord >= Me.Recordset.RecordCount Then
        Me.cmdNext.Enabled = False
    Else
        Me.cmdNext.Enabled = True
    End If
    If Me.NewRecord Then
        Me.cmdNew.Enabled = False
    Else
        Me.cmdNew.Enabled = True
    End If
End Sub
```

The code ensures that when the form is showing the first record the *Back* button is disabled; when showing the last record the *Next* button is disabled; when showing a new record both the *Next* and *New* buttons are disabled; and when showing any other record no buttons are disabled (*Fig. 18*).

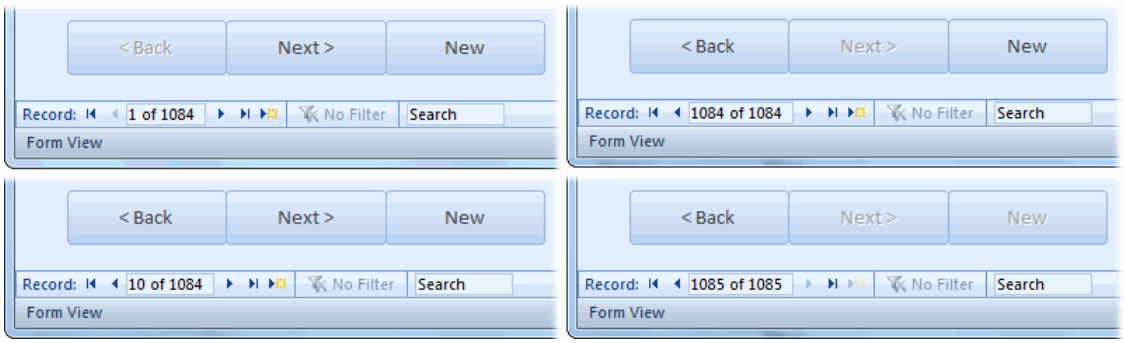

Fig. 18 Buttons are enabled or disabled according to which record is showing.

If you have chosen to add First and Last buttons, remember to add code to their **On Click** event as you did for the other buttons, using **acFirst** and **acLast** as appropriate.

First and Last buttons will also require some additional code in the **Form_Current** event if they are to be enabled or disabled according to which record is being shown. Modify the code so the resulting event procedure looks like this (*Listing 5*):

Listing 5:

```
Private Sub Form_Current()
    On Error Resume Next
    If Me.CurrentRecord = 1 Then
        Me.cmdBack.Enabled = False
        Me.cmdFirst.Enabled = False
    Else
        Me.cmdBack.Enabled = True
        Me.cmdFirst.Enabled = True
    End If
    If Me.CurrentRecord = Me.Recordset.RecordCount Then
        Me.cmdLast.Enabled = False
    Else
        Me.cmdLast.Enabled = True
    End If
    If Me.CurrentRecord >= Me.Recordset.RecordCount Then
        Me.cmdNext.Enabled = False
    Else
        Me.cmdNext.Enabled = True
    End If
    If Me.NewRecord Then
        Me.cmdNew.Enabled = False
    Else
        Me.cmdNew.Enabled = True
    End If
End Sub
```

When the form is showing the first record the *First* button is disabled; when showing the last record the *Last* button is disabled. Both buttons are enabled at all other times.

Coding Notes

If you are new to writing VBA code these notes should help you understand what the code in this project does and how it works.

Event Procedures

Much of the code associated with Access forms takes the form of *Event Procedures*. A *Procedure* is just another term describing a self-contained collection of VBA commands. Other terms commonly used are *Macro* and *Subroutine* or *Sub*. The procedures used here are *Event Procedures* meaning they run automatically when a particular event happens (or "fires" as we propellorheads like to say!).

The event procedures used in this project are the *On Click* event of each of the command buttons and the *On Current* event of the form itself. The various parts of a form and its controls usually have many different types of event for which a procedure can be created. To see a list of what is available, select an item on a form in Design view and take a look at the Event tab of the Property Sheet.

The Visual Basic Editor automatically names event procedures, the names being a combination of the object's name and the type of event, separated by an underscore, for example *cmdBack_Click* or *Form_Current*.

Handling Errors

These event procedures all begin with an *Error Handler*. This tells Access what to do if something goes wrong. Error handlers are often quite detailed, instructing Access how to proceed for each possible error that might happen and what to do if an unforeseen error occurs. They usually also attempt to rescue the situation if possible and, most importantly, try to prevent the program itself crashing.

Here, a very simple error handler is used, `On Error Resume Next`, which tells Access simply to ignore any error that might occur and to proceed to the next command. This kind of error handler should be used with caution since it is often not safe simply to ignore a problem. In these examples however its use is perfectly safe and there is no need to create anything more specific.

The Button Click Events

The `DoCmd` method offers access to a wide range of commands in Access and mirrors much of what is available in Access Macros. Here we have used `DoCmd.GoToRecord` which instructs Access to

move a specific data object, in this case our form, to a particular place in its recordset. It allows you so specify the type and name of the data object in question but since we are dealing with the current object those parameters can be left empty and only the parameter detailing where to go need be supplied. All of the button click events work the same way (*Listing 6*).

Listing 6:

```
Private Sub cmdBack_Click()
    On Error Resume Next
    DoCmd.GoToRecord , , acPrevious
End Sub
```

As usual the Visual Basic Editor helpfully supplies a list of options to choose from.

The Form Current Event

As explained earlier, it is always better to try to avoid an error than to deal with one that has already happened. For this reason a set of commands was created to disable those buttons whose function might have caused a problem if they were clicked at a particular time. For example, if the user was to click the Back button when on the first record.

Since the combination of available buttons changes according to which record is being displayed, the best time to run a procedure that checks and then enables or disables buttons, is when the form moves from one record to another. The form's *Current* event is suitable for this because it fires whenever a new or different record is displayed. This happens when a form opens, when the user moves from record to record, and when the user asks to create a new record.

The code used here consists of a number of *If Statements*. In VBA an If Statement can take a number of different forms. Here, each If Statement consists of a condition and two parts, what to do if the condition evaluates to *True* and what to do if it evaluates to *False*.

The code uses the keyword `Me` to refer to the current form. The form's `CurrentRecord` property returns a number which represents the index number of the current record in the form's recordset. So, for the first record it returns 1, the second record 2 and so on. If the form's `CurrentRecord` is 1 then it is displaying the first record so the first If Statement disables the *First* and *Back* buttons, otherwise it enables those buttons (*Listing 7*).

Listing 7:

```
    If Me.CurrentRecord = 1 Then
        Me.cmdBack.Enabled = False
        Me.cmdFirst.Enabled = False
    Else
        Me.cmdBack.Enabled = True
        Me.cmdFirst.Enabled = True
    End If
```

The second If Statement checks to see if the form is displaying the last record. But since the number of records in the recordset is likely to change we have to ask if the index number of the displayed record is the same as the number of records in the recordset. The If Statement uses the `RecordCount` property of the form's `Recordset` property to ascertain the total number of records and compares it with the `CurrentRecord` to decide whether or not to enable the *Last* button (*Listing 8*).

Listing 8:

```
    If Me.CurrentRecord = Me.Recordset.RecordCount Then
        Me.cmdLast.Enabled = False
    Else
        Me.cmdLast.Enabled = True
    End If
```

The If Statement for the *Next* button is very similar but instead of asking if the `CurrentRecord` *equals* the `RecordCount` it asks if the `CurrentRecord` is *greater than or equal to* the `RecordCount`. This is because if it is possible to move to a new record by using the Next command from the last record in the recordset. This effectively creates a value for the `CurrentRecord` of 1 greater than the number of existing records (e.g. record number 1085 of 1084 records!). (*Listing 9*)

Listing 9:

```
If Me.CurrentRecord >= Me.Recordset.RecordCount Then
    Me.cmdNext.Enabled = False
Else
    Me.cmdNext.Enabled = True
End If
```

Finally we make use of the form's `NewRecord` property which returns `True` if a new, unsaved record is being displayed. Note that, when using a logical condition (one that could be either True or False) it is not always necessary to specify *=True*. The statement `If Me.NewRecord` means the same as `If Me.NewRecord = True`. (*Listing 10*)

Listing 10:

```
If Me.NewRecord Then
    Me.cmdNew.Enabled = False
Else
    Me.cmdNew.Enabled = True
End If
```

Summary

This concludes this Masterclass. In it you learned how to improve the database user's experience by adding some useful buttons to an Access form for easily navigating through the form's records. You were shown different ways to accurately specify the size and placements of objects on a form and how to give them logical and meaningful names. Finally you added VBA code to power the buttons, including some useful refinements along the way.

All the techniques described here have been tested successfully in Microsoft Access versions 2007, 2010 and 2013.