VBA Made Easy

Access VBA Fundamentals

Level 1

www.AccessAllInOne.com

This guide was prepared for AccessAllInOne.com by:
Robert Austin

This is one of a series of guides pertaining to the use of Microsoft Access.

Contents

Introduction

Assumptions

We assume the following:

- You have a working knowledge of Microsoft Access (2007 or 2010).
- You know how to create Tables, Queries, Forms, Reports and Macros.
- You know how to add Controls to Forms and Reports.

Terminology

- Controls will refer to objects such as text-boxes, combo-boxes and list-boxes.

01 - The VBA Editor, Converting Macros

Learning Objectives

After you have finished reading this unit you should be able to:

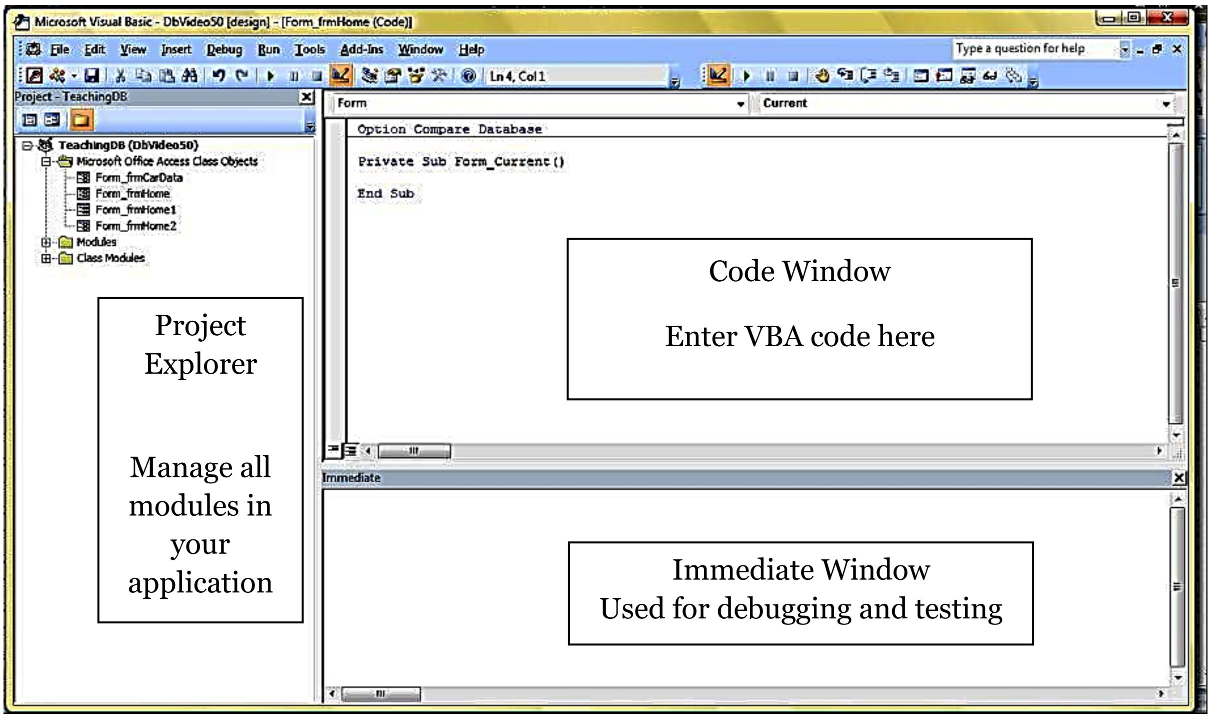
- Open the VBA editor in a number of different ways.
- Identify the code window, project explorer and immediate window.
- Select different forms and reports from the project explorer.
- Select and rename modules.
- Use basic tools for writing code.
- Understand the “DoCmd” object.
- Convert Macros to VBA code.

Introduction

The VBA Editor is what we use to enter VBA code for Forms, Reports, custom functions and more. In fact all Microsoft Office applications (Word, Powerpoint, Excel, Outlook) use the same VBA Editor, so although we are learning to program with Access you will also be able to transfer ALL your skills and knowledge and even automate between applications.

Note
In this unit you will be seeing many examples of code in order to demonstrate some of the features of the VBA Editor. You are not expected to understand everything and some of it may even seem particularly complicated. But rest assured! We will be covering everything in detail throughout this course.

The first part of the unit will involve getting to grips with the VBA editor and understanding the functions of the various windows



```
Option Compare Database

Private Sub Form_Current()

End Sub
```

Figure 1.1

Figure 1.1 is the VBA Editor with three areas highlighted; the Project Explorer, Code Window and Immediate Window. This is what is known as an Integrated Development Environment, which means everything you need to write programs and code are all in this one window.

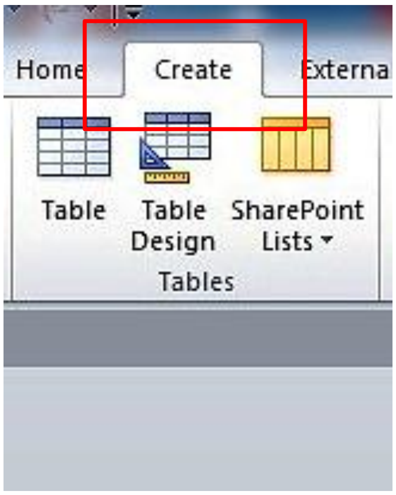
Note
The IDE is quite a simple idea but the first IDEs only arrived in 1995! Before that time, and this is still the case with some languages, the only way to compile programs was / is by using command-line tools.

More on the IDE in a few moments.

There are a couple ways to open the editor, all of which are quite natural once you learn them.

The VBA Editor through the Ribbon

From the ribbon select Create tab and to the far right is the Macro drop down; select Module. Now you carry out the same sequence of commands and open the VBA Editor.



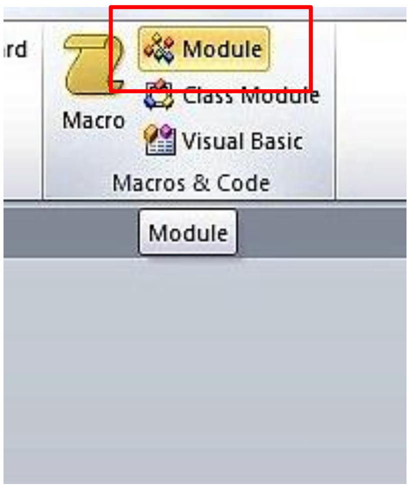


Figure 1.2

VBA Editor through the Form Designer

When in the form designer you can click the VBA Editor Button under Tools to bring up the IDE.

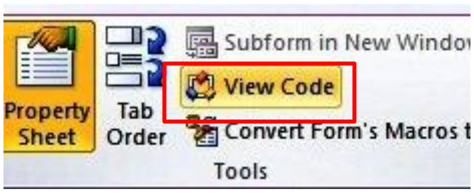


Figure 1.3

VBA Editor through the Form Designer Properties Window

Or, if you open the property window (F4) and click the Events tab, any of the ellipses (...) will open the VBA editor. The form must be in design view, however.

We suggest you do this as an exercise now. It will probably be the most common way you open the VBA editor.

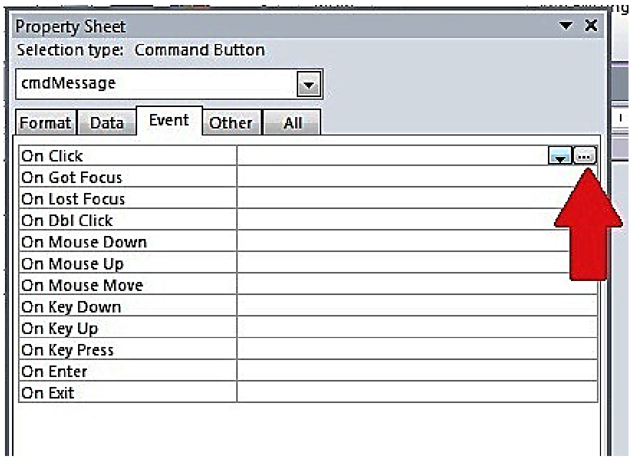
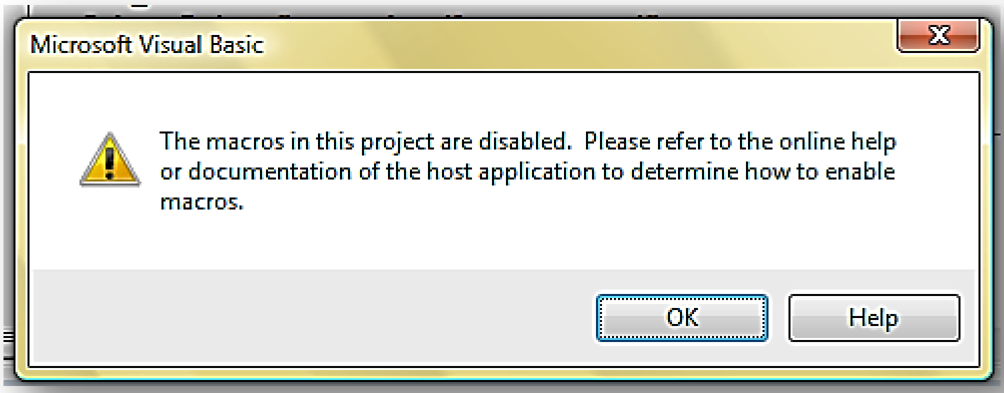


Figure 1.4

VBA Code not Working - Activating VBA Code

If you ever see this dialog:

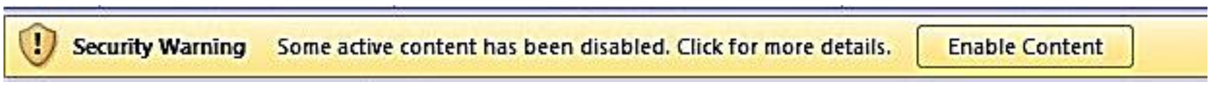


Microsoft Visual Basic

The macros in this project are disabled. Please refer to the online help or documentation of the host application to determine how to enable macros.

Figure 1.5

It is probably because you need to activate this:



Security Warning Some active content has been disabled. Click for more details.

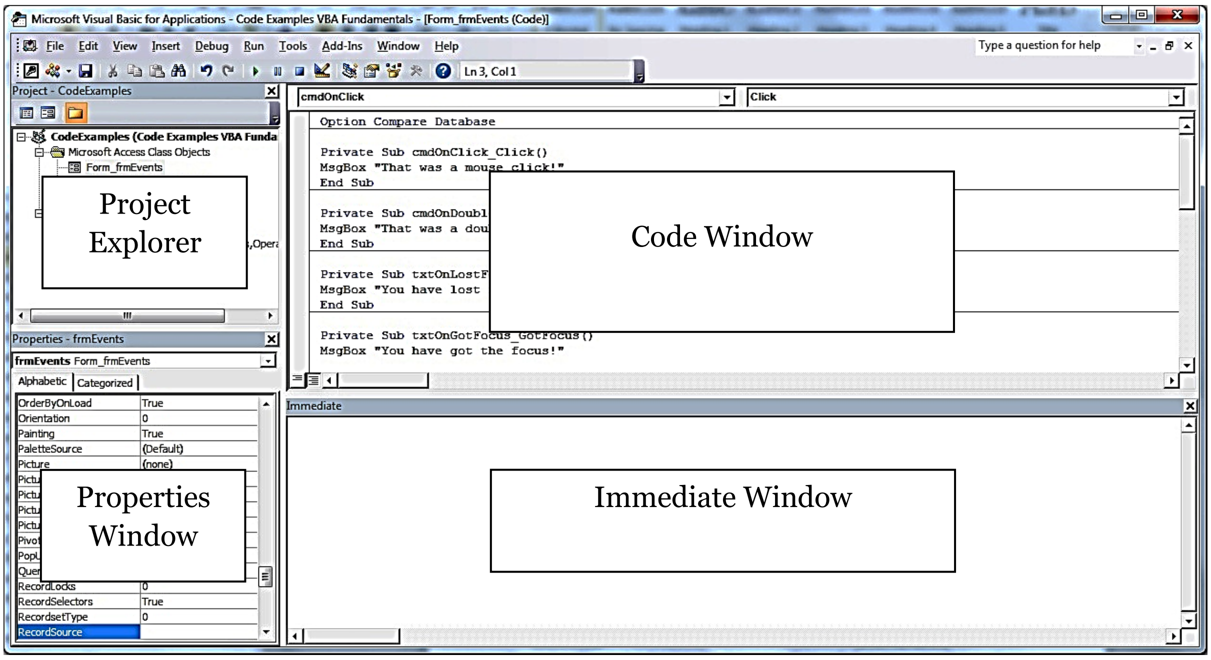
Figure 1.6

Navigate back to Access front end.
Click on Options... and tick the <u>Enable Content</u> Checkbox. Done.

Note
This is a feature of Access 2007/2010 which disables all VBA code until explicitly allowed to function. You can tell Access not to display this error but you need to set up a "Trusted Location", which is basically a nominated area where you can place programs that you know to be safe.

The VBA Editor Explained

There are five main areas of the editor that you need to know about. Here are four:


Microsoft Visual Basic for Applications - Code Examples VBA Fundamentals - [Form_frmEvents (Code)]
File Edit View Insert Debug Run Tools Add-Ins Window Help
Type a question for help
Ln 3, Col 1
Project - CodeExamples
CodeExamples (Code Examples VBA Funda
Microsoft Access Class Objects
Form_frmEvents
Project Explorer
Properties - frmEvents
frmEvents Form_frmEvents
Alphabetic
Categorized
Properties Window
cmdOnClick
Click
Code Window
Immediate
Immediate Window

```
Option Compare Database

Private Sub cmdOnClick_Click()
MsgBox "That was a mouse click!"
End Sub

Private Sub cmdOnDoubl
MsgBox "That was a dou
End Sub

Private Sub txtOnLostF
MsgBox "You have lost
End Sub

Private Sub txtOnGotFocus_GotFocus()
MsgBox "You have got the focus!"
```

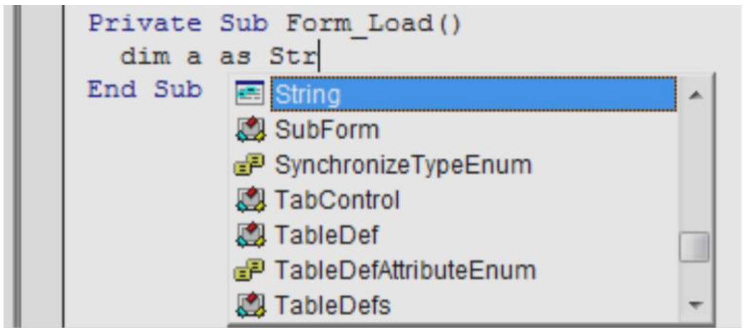
OrderByOnLoad	True
Orientation	0
Painting	True
PaletteSource	(Default)
Picture	(none)
RecordLocks	0
RecordSelectors	True
RecordsetType	0
RecordSource	

Figure 1.7

Code Window

The Code Window is where all your VBA code will be written. It has syntax highlighting, which means keywords in VBA, - such as Function, CStr, Return and others - will all appear in one colour, numbers in another colour, punctuation, comments and strings in yet other colours. It looks more appealing and makes reading lines of code much easier.

Another feature of the editor is called Code Completion. This is a useful feature; when you type in commands the editor will display possible values which it believes you may need. For example if you type **Dim a As Str**_ this ---------> will happen.



```
Private Sub Form_Load()
  dim a as Str
End Sub
```

Figure 1.8

Project Explorer Tree

The project explorer shows you all the modules available in your database and any add-ins or libraries you've included.

Modules are kept in three areas:

* Microsoft Office Access Class Objects

* Standard Modules

* Class Modules

Select a form module and double click it to see any sub procedures or functions it may contain.

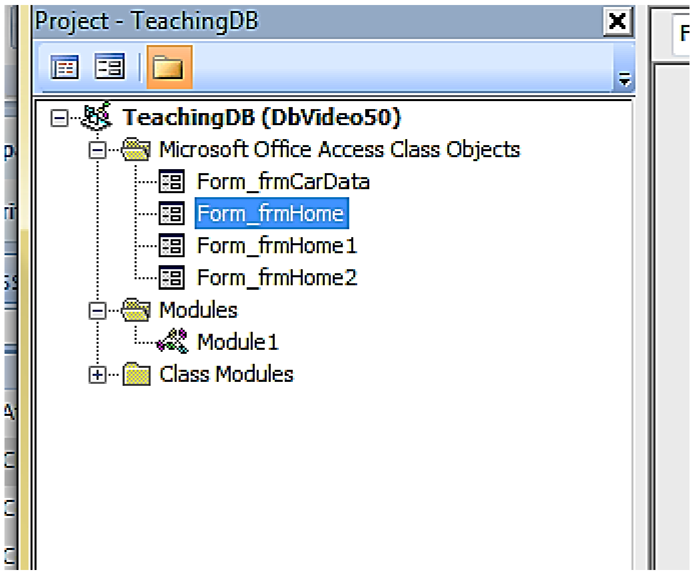


Figure 1.9

Microsoft Office Access Class Objects

These are VBA modules that are owned by (or children of) Forms and Reports. Their name will always be prefixed with "Form_" or "Report_". In these modules you will put all the event code that makes your forms and reports perform essential actions – like opening and closing. Unlike Standard Modules code in these modules is not normally available outside the form, they are private.

Now, using the Teaching Institute Software System, open one of the form modules and look at the sub procedures (Sub in VBA language). All of them start with **Private** which means the only code within that module can use the methods.

Standard Modules

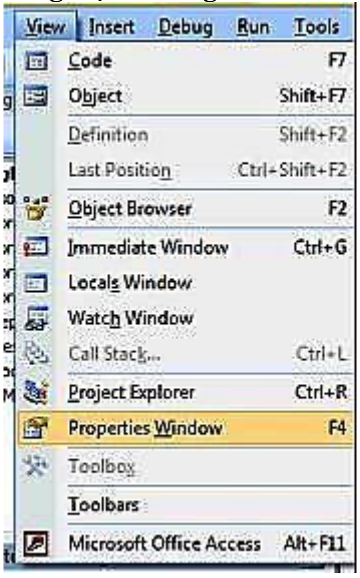
Standard modules contain code which may be accessed by any of the module types. In Standard Modules will go code that doesn't belong in forms or reports, for example a library of business rules or constants and types which are used by Forms and Reports. By default anything written in here is available anywhere else in the project – this is known as global scope.

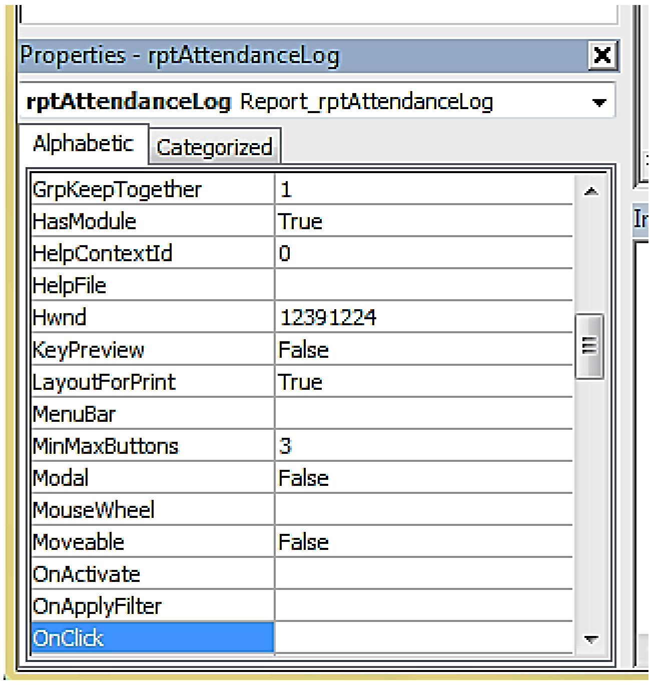
Class Modules

Each Class Module contains code that revolves around a Class, which is a type of data-type or object. By default, anything written here is available elsewhere in the project. Just to mention here that Class Modules ultimately support the Standard Modules and Access Class Object Modules by providing new functionality not supplied with Access and specialist functionality that you have developed and your organisation may need in other projects.

Properties Window

The properties window is available in two places; the first is in the Form Design window docked to the right. The second is usually bottom left in the VBA IDE. If it's not there you can bring it up by pressing F4 or using the view menu.





Properties - rptAttendanceLog

rptAttendanceLog Report_rptAttendanceLog

Alphabetic | Categorized

GrpKeepTogether	1
HasModule	True
HelpContextId	0
HelpFile	
Hwnd	12391224
KeyPreview	False
LayoutForPrint	True
MenuBar	
MinMaxButtons	3
Modal	False
MouseWheel	
Moveable	False
OnActivate	
OnApplyFilter	
OnClick	

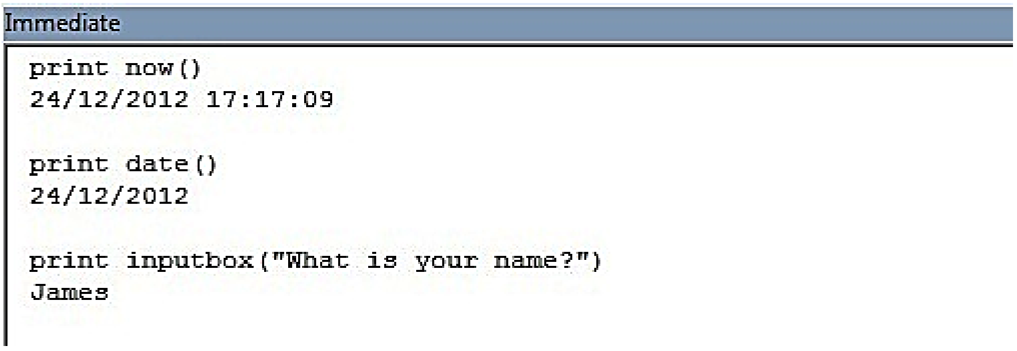
Figure 1.10

Immediate Window

The immediate window is located at the bottom of the screen and is the big blank window that says immediate in the title.

Using this window, you can test code snippets, test out your functions directly rather than through a form's button, and also debug your code.

Here is a small introduction. Try typing **print now()** and see what date and time come up. Then try **Print InputBox("what's your name?")**



Immediate

```
print now()
24/12/2012 17:17:09

print date()
24/12/2012

print inputbox("What is your name?")
James
```

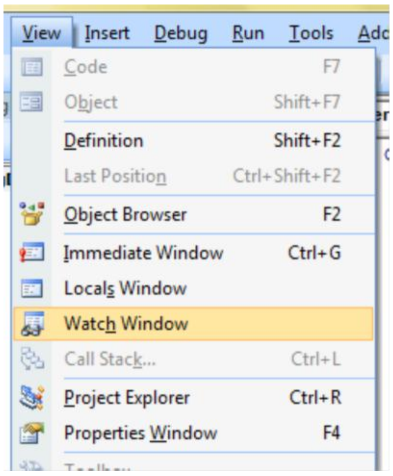
Figure 1.11

Watch Window

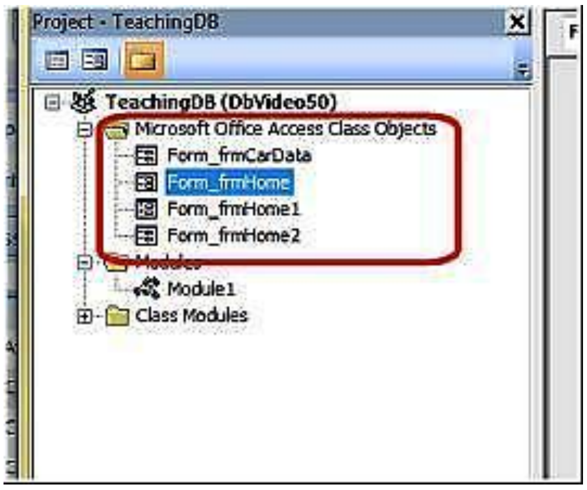
Figure 1.12

It is a little premature to bring in the Watch Window but you will be using it soon so just take note.

The Watch Window is used in debugging your code to watch and keep an eye on the variables in your code. You can also set a trap, so if one of your variables becomes a particular value you can trigger a break which will cause your program to stop running so you can inspect its state. More on this topic in the next unit.



How to select different Forms and Reports (from project explorer)



In the Project Explorer the items contained within folders are all Modules.

Here we have 4 Form modules and 1 Standard module.

If you select and double click a form, the VBA module code will load up.

Ultimately all forms will have a module of their own because forms look pretty but do little without VBA code.

You can explore this now by opening a form that doesn't have a module and giving it one. There are a few ways to do that but here's the most straightforward.

- Open a form in design view.
- Click on a control, like a button.
- In the Properties window click on the Events tab.
- Click the Ellipsis of the On Click event.
- Access will now automatically create a new module with the name **Form_myForm**!

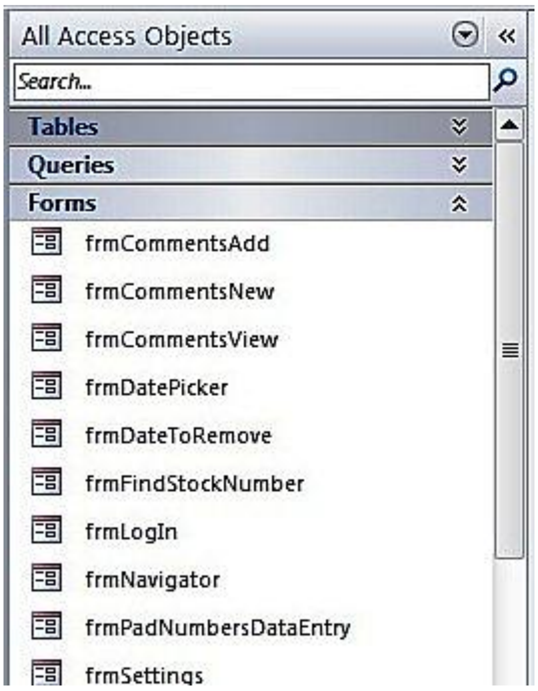


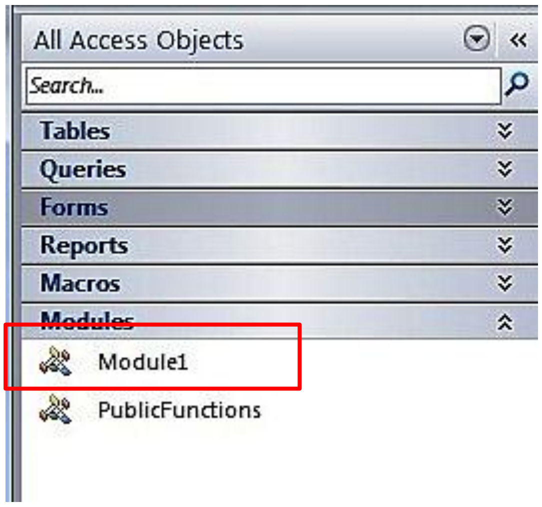
Figure 1.13

How to select different Modules

Modules can be seen in two places, in the Access IDE and the VBA Editor IDE. Modules in the Access IDE are visible because they have no owner, per say, except that they belong to the project you are working on. Modules that "belong" to forms and reports are not shown here; they show up in the VBA Editor IDE.

If you want to add new modules use the instructions shown previous.

Access IDE



VBA Editor IDE

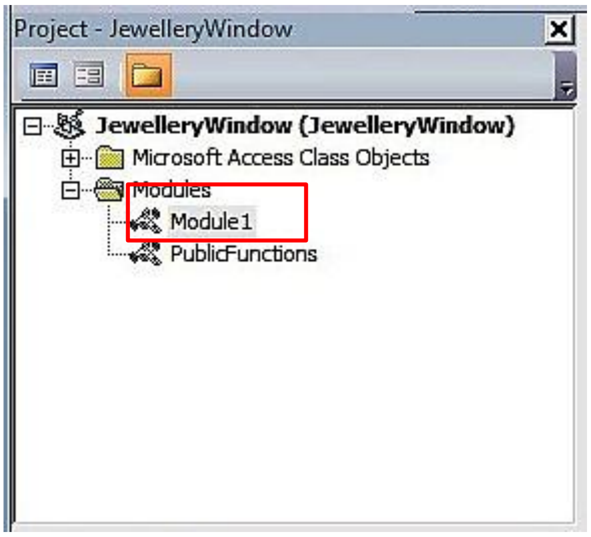
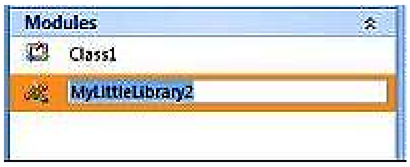

Project - JewelleryWindow
JewelleryWindow (JewelleryWindow)
Microsoft Access Class Objects
Modules
Module1
PublicFunctions

Figure 1.14

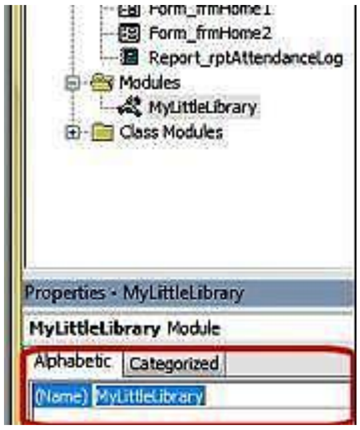
How to Rename Modules

Unlike other objects in Access renaming modules is easy, even when they are open.



Select Module and hit F2---^^^
Or

Select the module in the VBA IDE and change the property (Name)------------------------------>



(Name) MyLittleLibrary

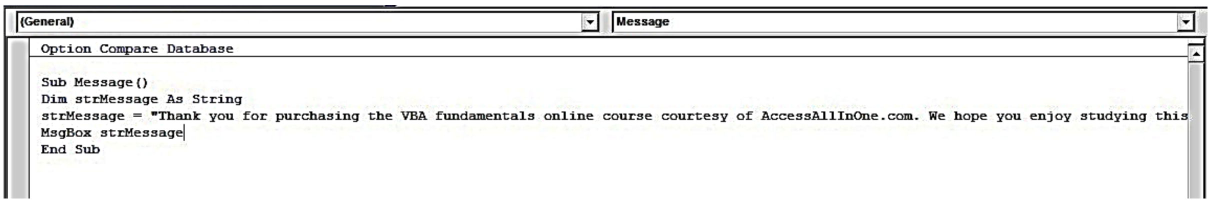
Figure 1.15

Basic Tools for Writing Code

The VBA Editor incorporates a number of useful features which help you whilst you are developing, testing and in production (some of which we have already touched on). Here we'll take a closer look at a few code writing features of the VBA editor.

Line Continuation Character

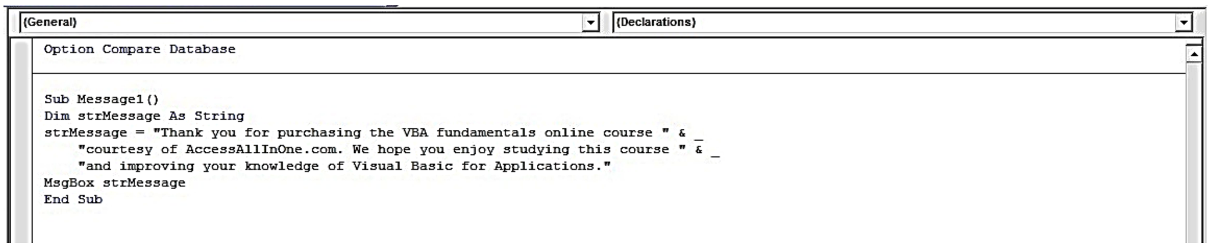
When we write code, we are often required to create string expressions that are wider than the page itself! Although the entire string is on one line, it makes it more difficult to code as you constantly have to use the horizontal scroll bar to read exactly what is written.



```
Option Compare Database

Sub Message()
Dim strMessage As String
strMessage = "Thank you for purchasing the VBA fundamentals online course courtesy of AccessAllInOne.com. We hope you enjoy studying this
MsgBox strMessage
End Sub
```

Figure 1.16

The smart people at Microsoft came up with a simple plan, the line continuation character, or space and underscore for short. We use this with an ampersand (&) to make our code easier to read.

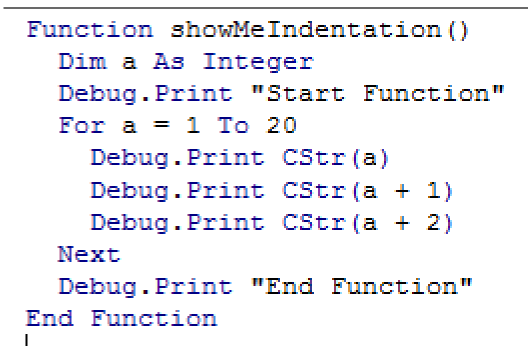


```
Option Compare Database


Sub Message1()
Dim strMessage As String
strMessage = "Thank you for purchasing the VBA fundamentals online course " & _
    "courtesy of AccessAllInOne.com. We hope you enjoy studying this course " & _
    "and improving your knowledge of Visual Basic for Applications."
MsgBox strMessage
End Sub
```

Figure 1.17

Indenting Code

Another key assistant to reading our code is the indentation. Indentation gives us a clear indication of code blocks. Indentation is implicit, so typing in a Public Function Name() and pressing the enter key will add the End Function and indent your code by two or four spaces or a tab. This is basically standard practice across all programming languages.



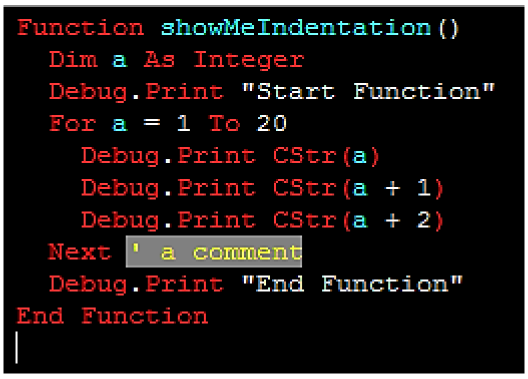
```
Function showMeIndentation()
  Dim a As Integer
  Debug.Print "Start Function"
  For a = 1 To 20
    Debug.Print CStr(a)
    Debug.Print CStr(a + 1)
    Debug.Print CStr(a + 2)
  Next
  Debug.Print "End Function"
End Function
```

As you can see, it's easy to see which bits of code are associated with one another.

Figure 1.18

Editor Format to Adjust Colours

The above example also neatly brings us to syntax highlighting. All keywords in VBA are dark blue by standard and anything we write is in a black font (except comments which are green). Syntax highlighting serves to use our sense of colour to add meaning to the code. You can change the colours too! Follow the steps below: Click on the Tools drop down menu, select Options... tab over to Editor Format and change as needed. Working under poor lighting conditions makes the default black on white very uncomfortable, so perhaps a blackened theme would be more suitable.



```
Function showMeIndentation()
  Dim a As Integer
  Debug.Print "Start Function"
  For a = 1 To 20
    Debug.Print CStr(a)
    Debug.Print CStr(a + 1)
    Debug.Print CStr(a + 2)
  Next ' a comment
  Debug.Print "End Function"
End Function
```

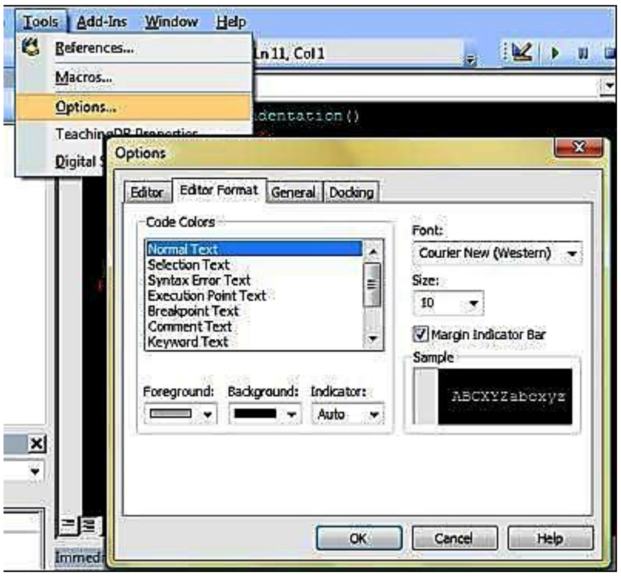


Figure 1.19

Naming Conventions

A naming convention is a way of naming variables and objects which your developer group uses to make code you develop easier to understand. In Figure 1.19, Function showMeIndentation() is in “camelCase “. Here are some additional naming convention details:

- Use meaningful variable names – make your variables mean something. Zzxd isn’t meaningful, but fileNotFound has a semantic meaning for humans (even though it doesn’t affect the computer or VBA in any way).
- Use camelCase for variables and functions – that is, for every word in your variable name make the first letter upper-case, except the first letter of the first word. thisIsCamelCase()
- Use CamelCase for classes, and types – capitalise your enumerations, user-defined types, class name, but leave functions and variables in camelCase.
- Use UPPER_CASE for constants – when you declare a constant the name of that constant is usually capitalised. This means nothing to the compiler but means everything to you.

Amongst VBA developers a typical naming convention is to start all variables with a letter indicating the variable type (in lower case):

- iMyNumber would be of type Integer
- dblMyOtherNumber would be of type Double
- sText would be of type String

With form and report controls a three letter prefix is very common also:

- txtMyTextBox
- cboMyComboBox
- lblLabel

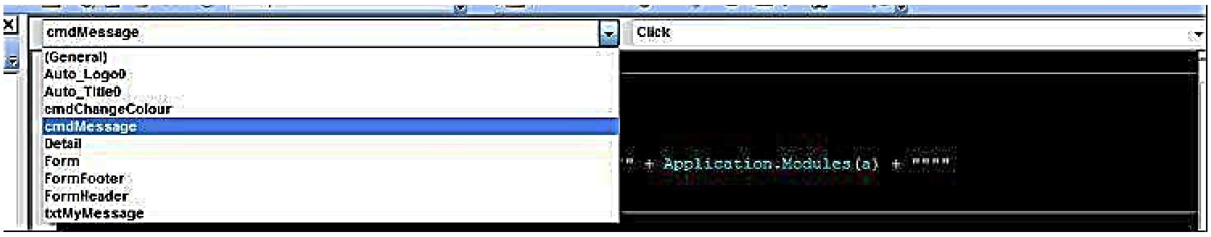
But, the point of a naming convention is to make your code more accessible to others by imposing on you and your colleagues a consistent way of writing code. Feel free within your departments or projects to use whatever naming convention you like, but the key is be consistent.

If you are working in a large group of developers consider the following:

- keep a module which bears your name and contains the functions for which you are responsible
- prefix all your functions and subroutines with your initials – the idea here being that your procedures and function do not clash with other developers

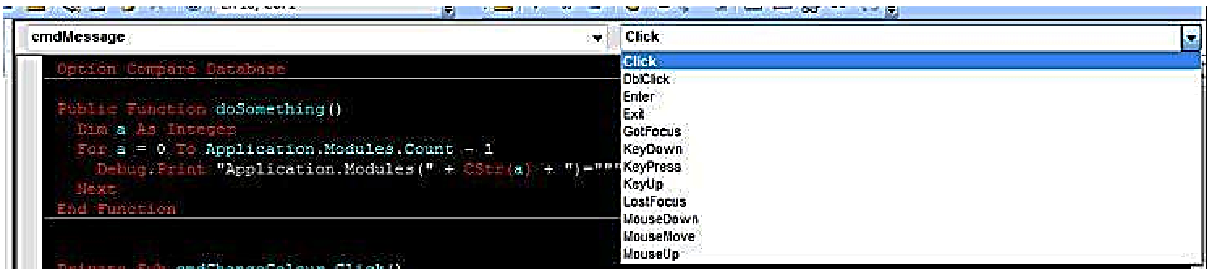
Select Object Dropdown and Procedures Dropdown

Hidden in plain view are two drop down menus just above the Code Editor Window. The drop down menu on the left is used to select and even indicate which control your cursor is currently in and the one on the right lists all available events for that object.



```
" + Application.Modules(a) + """"
```

Figure 1.20

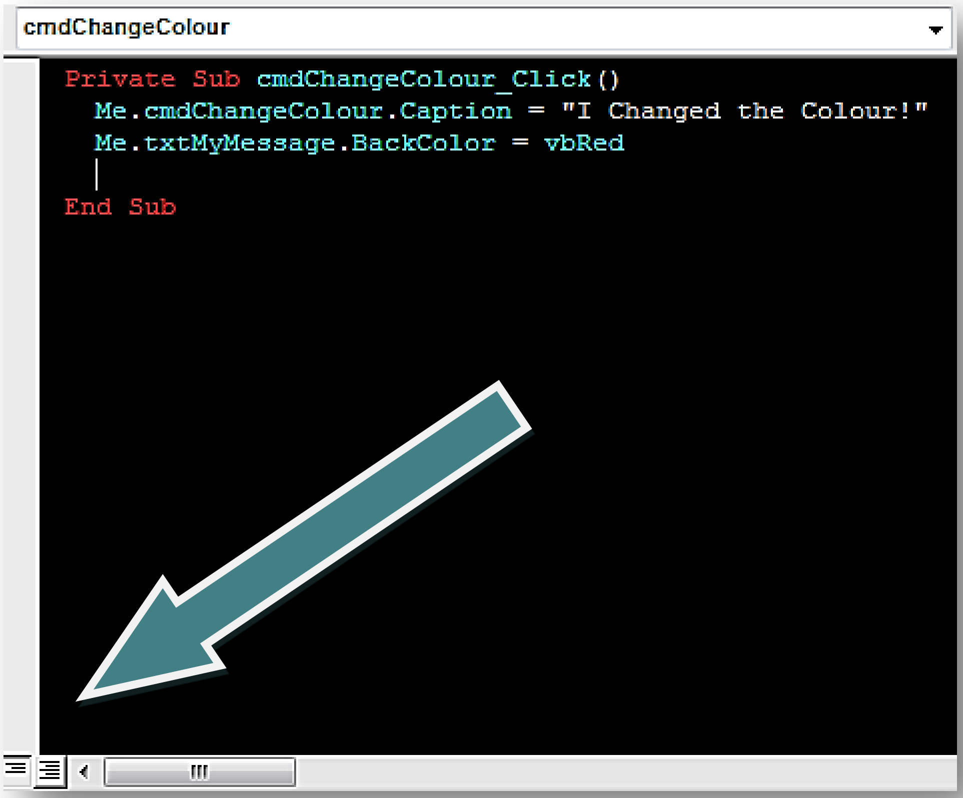


```
Option Compare Database

Public Function doSomething()
  Dim a As Integer
  For a = 0 To Application.Modules.Count - 1
    Debug.Print "Application.Modules(" + CStr(a) + ")="""
  Next
End Function
```

Figure 1.21

Procedural View and Full Module View

Another useful feature is the ability to switch between module and procedure view in the code window. Module view is what we normally see but if you have lots of procedures and functions and would like to only view the one you are working on, click the button on the left in Figure 1.22 . All the other code will magically disappear!



```
Private Sub cmdChangeColour_Click()
  Me.cmdChangeColour.Caption = "I Changed the Colour!"
  Me.txtMyMessage.BackColor = vbRed

End Sub
```

Figure 1.22

DoCmd Syntax and Arguments Explained

One of the most versatile objects in Access VBA is DoCmd. This one object gives you the programmer control over how your application operates, issues orders and gives you access to features that are otherwise confined to the GUI.

For example: on a form you have a button with the caption “Close”. Clicking it does nothing, because we haven’t added code. So, open a form, add a button control, tab over to Events and click OnClick. Now enter the following code into the procedure:

```
    DoCmd.Close ‘ closes the presently active window

    DoCmd.Close acForm, Me.Name    ‘ Closes any windows with the
                                   ‘ Name of Me.Name
```

Figure 1.23

If you use DoCmd.Close it does the obvious, almost. If placed in a form’s OnClick Event the DoCmd object will fire but it will fire first on the presently active window –which means if your user happens to be looking at a query result set that took 30 minutes to complete, you are going to have some explaining to do. So, where possible always qualify exactly what you want to do, as has been demonstrated in the previous code block example.

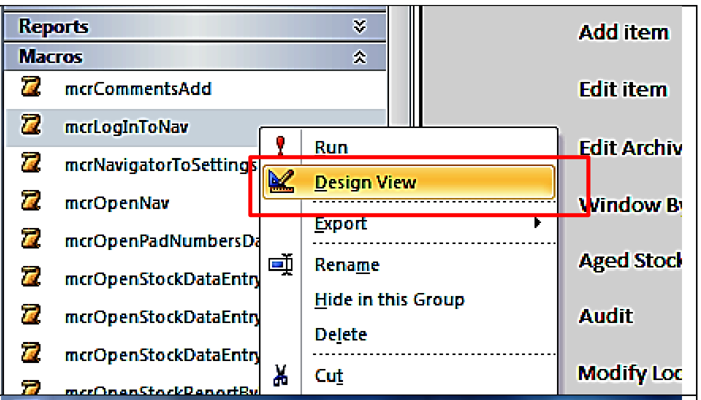
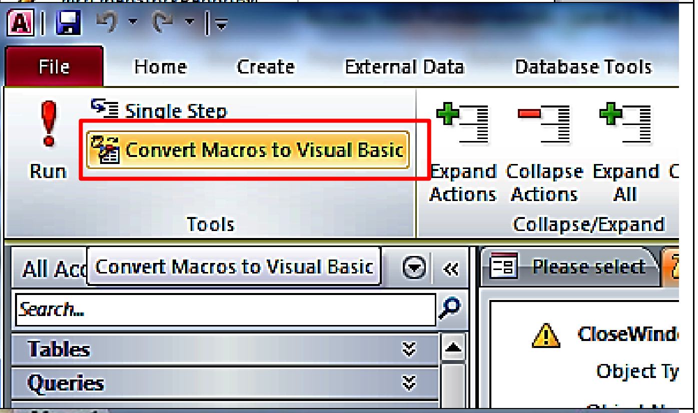
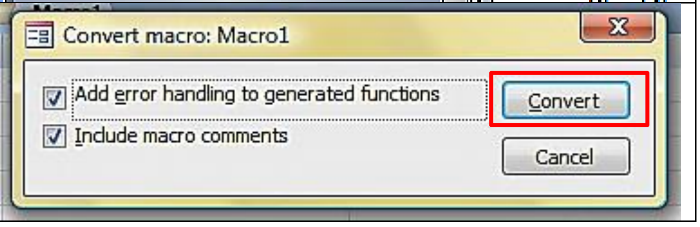
DoCmd has some other useful functions listened below.

```
DoCmd.OpenForm “name” ‘ does just that
DoCmd.MoveNext ‘ advances the form’s recordset cursor by one
DoCmd.FindFirst ‘ finds first record in database – may have been given search parameters
DoCmd.FindNext
DoCmd.Maximise
DoCmd.Minimise
DoCmd.RunCommand ‘ this one method has hundreds of commands relating to the GUI
DoCmd.Close acForm, Me.Name  ‘ closes any windows with the Name of Me.Name
DoCmd.RunSQL ‘ excellent for quickly executing an action query when Recordset setup would
take too long
```

Figure 1.24

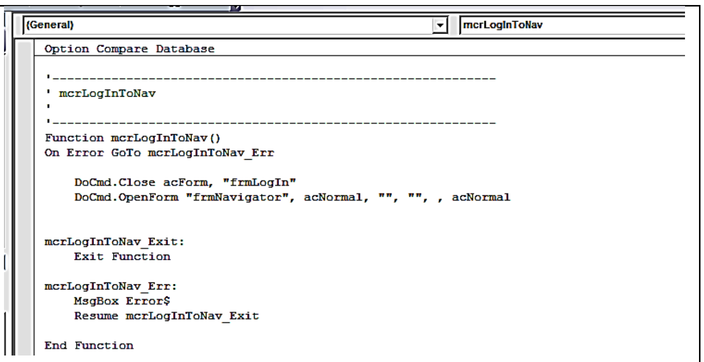
How to Convert a Standalone Macro

A Macro is an object that includes a list of instructions, and these instructions translate to VBA code almost directly. Rather than writing your code in VBA you could, and probably have already, put together a few Macros to close forms, open forms, email data, navigate records, etc.

Open a Macro in design view using the Navigation Pane.	 Reports Macros mcrCommentsAdd mcrLogInToNav mcrNavigatorToSettings mcrOpenNav mcrOpenPadNumbersDa mcrOpenStockDataEntry mcrOpenStockDataEntry mcrOpenStockDataEntry Run Design View Export Rename Hide in this Group Delete Cut Add item Edit item Edit Archiv Window B Aged Stock Audit Modify Loc
Click on Convert Macros to Visual Basic in the Tools group of the design tab of the ribbon.	 File Home Create External Data Database Tools Single Step Convert Macros to Visual Basic Run Tools Expand Actions Collapse Actions Expand All Collapse/Expand All Acc Convert Macros to Visual Basic Search... Tables Queries Please select CloseWind Object Ty
Click on convert.	 Convert macro: Macro1 Add error handling to generated functions Include macro comments Convert Cancel

When the conversion has finished a new Standard Module is created with all the code for the macro contained within.

Just like Macros the code in the Converted Macro module is available elsewhere in your project (unlike form generated code the functions do not have **Private** modifier before them (more on that in a future unit).



```
(General)                                   mcrLogInToNav
Option Compare Database

'------------------------------------------------------------
' mcrLogInToNav
'
'------------------------------------------------------------
Function mcrLogInToNav()
On Error GoTo mcrLogInToNav_Err

    DoCmd.Close acForm, "frmLogIn"
    DoCmd.OpenForm "frmNavigator", acNormal, "", "", , acNormal


mcrLogInToNav_Exit:
    Exit Function

mcrLogInToNav_Err:
    MsgBox Error$
    Resume mcrLogInToNav_Exit

End Function
```

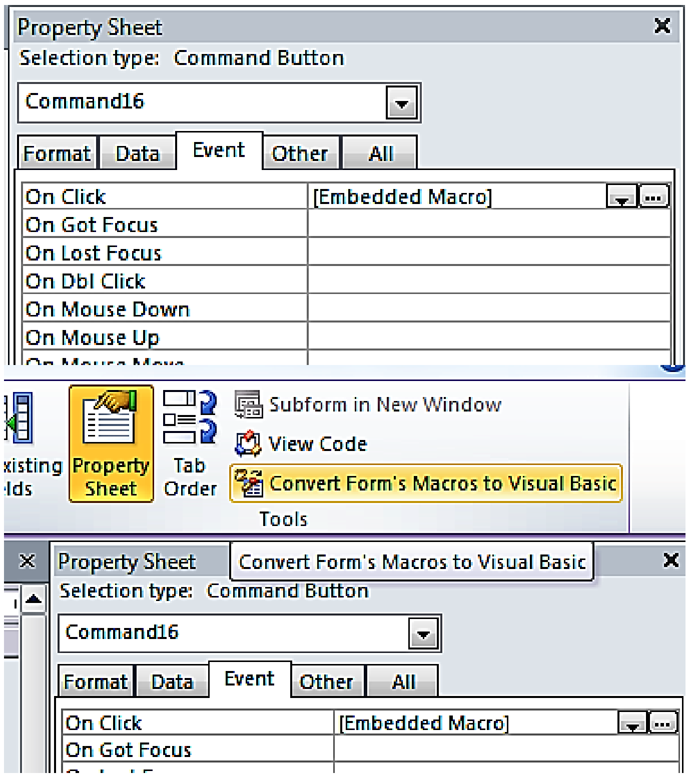
Figure 1.25

How to Convert a Form's Embedded Macros

Embedded Macros are children of their parent form. You will tell an embedded macro from an event procedure or normal macro because the Property Sheet of the Form Events tab give you [Embedded Macro].

Open the form in Design view (it won't work in Layout view).

Click on Convert Form's Macros to Visual Basic.



Property Sheet
Selection type: Command Button
Command16

On Click	[Embedded Macro]
On Got Focus	
On Lost Focus	
On Dbl Click	
On Mouse Down	
On Mouse Up	

Selection type: Command Button
Command16

On Click	[Embedded Macro]
On Got Focus	

Figure 1.26

When to use Macros and VBA

Macros are objects in the Access IDE that perform many application functions without needing to resort to code in VBA. VBA on the other hand does everything a macro can do and a whole lot more. Whether you choose to use a Macro or VBA will be down to your preference and familiarity with VBA or macros. In this course we are learning about VBA so we will exalt its merits over macros. But for many simple tasks macros are just fine.

Questions

1. Examine the following code. Will it do as the user expects? If not why not?

```
Sub Message()

Dim strMessage As String

strMessage = "What is your name?"

    MsgBox_
    strMessage ,_
    vbQuestion ,_
    "Name"
End Sub
```

2. Do the following and answer the questions below.

Create a new form called “Ex_2_Form1”.

Add a combo box, a textbox and a label.

   a. What is the third event in the properties window for combo box?
   b. On the label what is the third event property called?
   c. Does the textbox have a caption property?
   d. Select the Detail part of the form, click on Other. What is the name of the object?
   e. Select the form, open Other in property tab. What does “Cycle” mean?

3. True or false …
   a. Macros are VBA code?
   b. The Form Editor allows you to view and edit VBA code?
   c. The Property Sheet in Form View is the same as the Properties window in VBA IDE?
   d. IDE means Individual Data Execution?
   e. Macros have fewer commands than VBA code?
   f. The Immediate window allows code snippets to be tested and executed?
   g. A Class Module is a VBA symbol of elitism over other languages?
   h. The Watch window lets you watch the values of variables?

4. A module window is open, you have edited some code. You want to change the module’s name. How do you do this? (hint: there are two ways).

5. Indenting code is a form of what?
   a. Syntactic sugar.
   b. Readability aid.

c. Banging out code

6. Which desert animal features predominantly in variable naming conventions?

7. For the following prefixes what is the most likely variable or object type?
   a. txt
   b. i
   c. cmb
   d. lbl
   e. lng
   f. s
   g. frm
   h. qry
   i. mcr
   j. C

8. What must you do to convert a macro into a VBA function or procedure?

9. Which of the following are methods or variables that belong to the DoCmd object?
   a. Close
   b. RunSQL
   c. MimicForm
   d. Open
   e. SetWarnings
   f. Crash
   g. Beep
   h. RunSavedImportExport
   i. LockNavigationPane
   j. OpenHeadUpDisplay
   k. FindNextRecord

10. True or False? Code completion is a …
    a. …type of artificial intelligence.
    b. …VBA IDE feature.
    c. …developer assistant.
    d. …programming race.

11. Go to the Project Window and add two new Modules. What are their default names?

12. You are testing out some programming features and decide to use the Immediate Window to test your functions. What is wrong with the following code?

    a. Debug.Print (newFunctionTest 1)
    b. Print "This May "; " be my last chance " +_ "to get this right"
    c. Dim ab As String
    d. Ab = Array("a", 3)
       print Ab(2)

13. Why might the Project explorer window not have a form's module?

14. Open a form in design view and click back into the Navigation Pane Why can't you change the form's name?

15. The Line Continuation Character is a what?
    a. An actor.
    b. An overstatement.
    c. A space followed by an underscore.
    d. A VBA bloodline.

16. Procedural View or Full Module View? Which allows an overview of the code in the VBA IDE?

17. Embedded Macros are not available in the Navigation Pane. Why?

18. Why are Microsoft Office Access Class Objects Modules not visible in the Navigation Pane?

19. By default functions and procedures in a Standard Module are…
    a. Private
    b. Public
    c. Protected
    d. static

20. True or False? It is only possible to convert stand-alone macros in VBA.

02 - Objects, Properties and Methods

Learning Objectives

After you have finished reading this unit you should be able to:

- Say what an object is
- Say what a property is
- Say what a method is
- Say what a collection is
- Understand how to reference objects
- Understand how to reference properties

Objects

VBA is an object based language and can interact seamlessly with Access objects (along with objects from other Office programs such as Excel and Word). In the physical world objects are things like tables, cars and people and in the VBA world objects are things like Tables, Queries, Forms, Reports, RecordSets, Buttons, Combo-Boxes, List-Boxes, Text-Boxes, Charts, etc.

The secret to programming with an object based language is to understand how to manipulate these objects by taking advantage of their properties and methods.

Properties

Properties are said to be attributes of the objects they are attached to. As stated, command buttons are an object in VBA and contain various properties including the BackColor property, the Caption property, the Left property, the Top property and the ForeColor property. These are all attributes of the command button and help us to define various aspects of its appearance and behaviour.

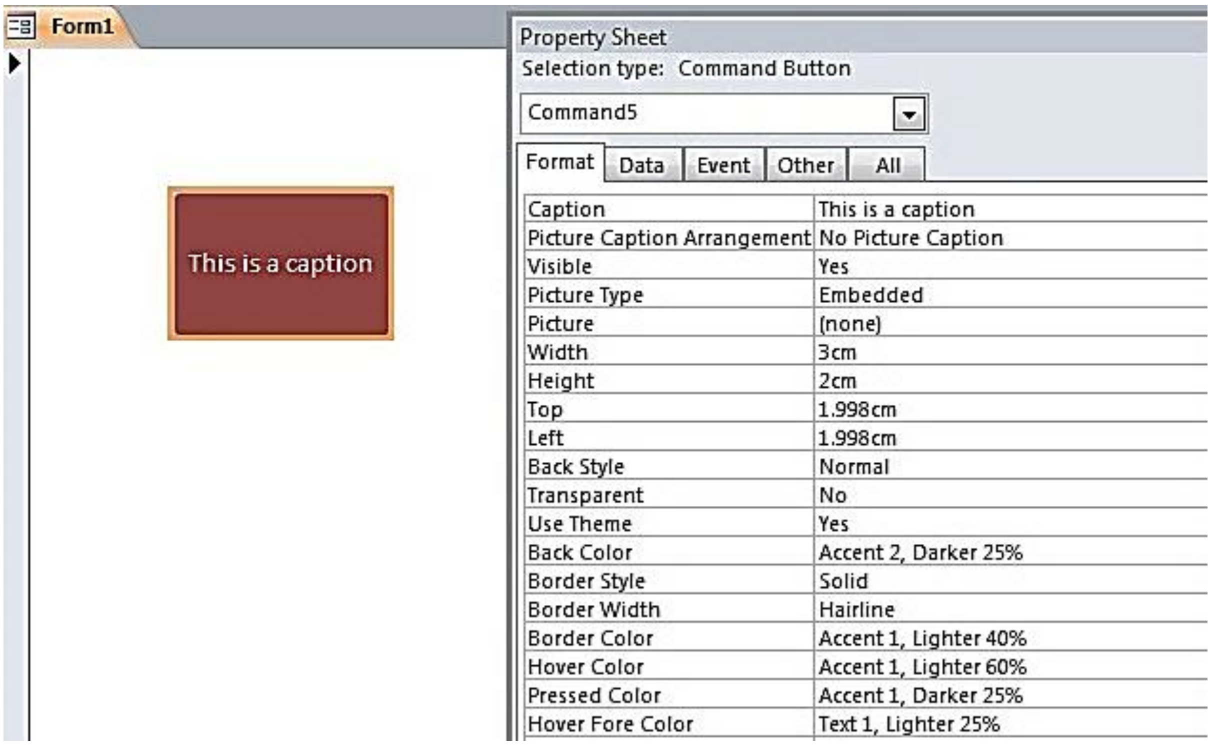
With the properties listed above we can:

- Determine the color of the Command button (BackColor property)
- Determine what text is displayed in the Command button (Caption property)
- Determine how far from the left of the form or report the Command Button lies (Left property)
- Determine how far from the top of the form or report the Command Button lies (Top property)
- Determine the color of the text in the Command Button (ForeColor property)

In Figure 2.1 a button has been added to a form. We have manipulated a few of the properties listed above and because of this we can say that:

- The Back-Color property is set to ***Accent 2, Darker 25%.***
- The Caption property contains the text ***This is a caption.***
- The Left property contains the value ***1.998 cm***

- The Top property contains the value ***1.998 cm***
- The ForeColor property contains the value ***Background 1***



Property Sheet
Selection type: Command Button

Command5

Format | Data | Event | Other | All

Caption	This is a caption
Picture Caption Arrangement	No Picture Caption
Visible	Yes
Picture Type	Embedded
Picture	(none)
Width	3cm
Height	2cm
Top	1.998cm
Left	1.998cm
Back Style	Normal
Transparent	No
Use Theme	Yes
Back Color	Accent 2, Darker 25%
Border Style	Solid
Border Width	Hairline
Border Color	Accent 1, Lighter 40%
Hover Color	Accent 1, Lighter 60%
Pressed Color	Accent 1, Darker 25%
Hover Fore Color	Text 1, Lighter 25%

Figure 2.1

Using the property sheet feel free to change the values of certain properties for the Command Button in order to see how they affect the appearance and behaviour.

Note
In Layout or Design view on a form or report, if you can’t see the property sheet go to the Design Tab in the Ribbon and then click the Property Sheet Button. (Alternatively, just press F4).

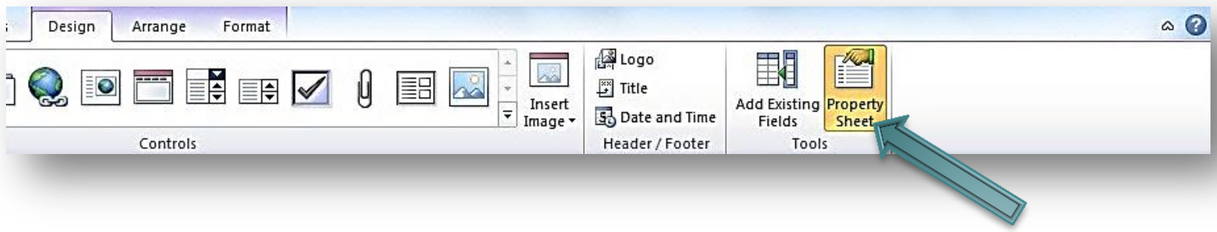


Figure 2.2

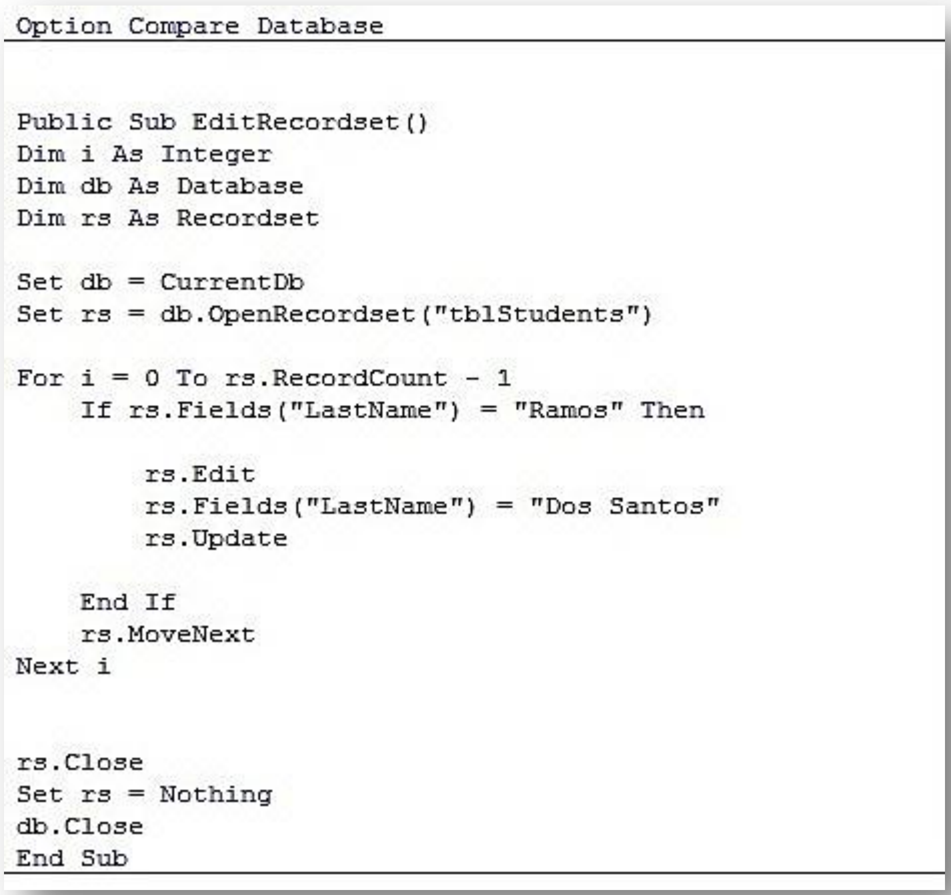
Methods

Methods can be described as actions that an object can perform. Objects such as Forms, Reports and Command Buttons don’t have a lot of methods that they can perform so we will be introducing something called the recordset object to give you a feel for methods.

The Recordset object

We will be dedicating a whole unit to the Recordset object later on in the course but for now we will be providing a brief overview of the Recordset object, its properties and crucially its methods.

The Recordset object allows us to open a table or query in memory (which means we can't see it) and add, modify and delete records. Used with a For Loop (explained in a later unit) it is a powerful tool to help us to manipulate data.



```
Option Compare Database

Public Sub EditRecordset()
Dim i As Integer
Dim db As Database
Dim rs As Recordset

Set db = CurrentDb
Set rs = db.OpenRecordset("tblStudents")

For i = 0 To rs.RecordCount - 1
    If rs.Fields("LastName") = "Ramos" Then

        rs.Edit
        rs.Fields("LastName") = "Dos Santos"
        rs.Update

    End If
    rs.MoveNext
Next i


rs.Close
Set rs = Nothing
db.Close
End Sub
```

Figure 2.3

In Figure 2.3 we open up a recordset for a table we have stored called "tblStudents". We now have that table definition in memory and can loop through each record and update it if we wish.

We are asking Access to locate the record for a student whose last name is "Ramos" and then change the last name to "Dos Santos".

In order to achieve this we have to use certain methods associated with the Recordset object including:

- Movenext – Moves to the next record.
- Edit – Ensures the record can be modified.

- Update – Writes any updates to the record.

These are all methods of the Recordset object and thus are defined as “actions that the recordset can perform”.

Note
We cover Branching, For Loops and the Recordset object in great detail later on in the course. Figure 2.3 is merely intended to provide you with an overview of how we use methods in VBA.

Collections

As we have stated forms, reports, queries and tables are all objects in an Access database. But normally we have more than one of each. We may have a table for students as well as a table to store courses, classes, teachers, etc. In the VBA environment we have the ability to refer to sets of objects of the same type using collections.

Collections are literally collections of objects of the same type (for example, there is a Forms collection and a Reports collection but not a Forms and Reports collection) and have their own properties and methods.

An example of a common property in a collection is the Count property. We use this to return the number of items in a collection. The Forms collection is highlighted in red.

```
Public Function IsLoaded(strFormName As String) As Boolean
Const conFormDesign = 0
Dim i As Integer
IsLoaded = False
For i = 0 To Forms.Count - 1
  If Forms(i).FormName = strFrmName Then
    If Forms(i).CurrentView <> conFormDesign Then
     IsLoaded = True
    Exit Function
   End If
  End If
Next
End Function
```

Figure 2.4

In Figure 2.4 we use the form collection and the count property to help us ascertain whether a particular form is loaded. We refer to the Forms collections as Forms and the Count property as Forms.Count.

Note
We cover Collections, Functions, For Loops and the If statement in great detail later on in the course. Figure 2.4 is merely intended to provide you with an overview of how we use collections in VBA.

Objects, Properties and Methods – An Analogy

Most people tend to understand that if you want to change the text in a command button you use the Caption property and if you want to change the back color you use the BackColor property. It almost seems like killing a fly with a nuclear bomb to explain objects, properties and methods when, in truth, they can be quite intuitive. There is, alas, an ulterior motive behind these machinations!

The concepts surrounding object based languages and programming mean that it is very important that programmers are not only able to use objects, properties and methods but are also able to understand the thinking behind them.

Note
Taking some time now to really get to grips with these ideas will pay dividends in the future. (You may have to trust us on that one for now!)

So, as with all courses that deal with objects, properties and methods we present an analogy! And for this particular analogy we shall be using a car!

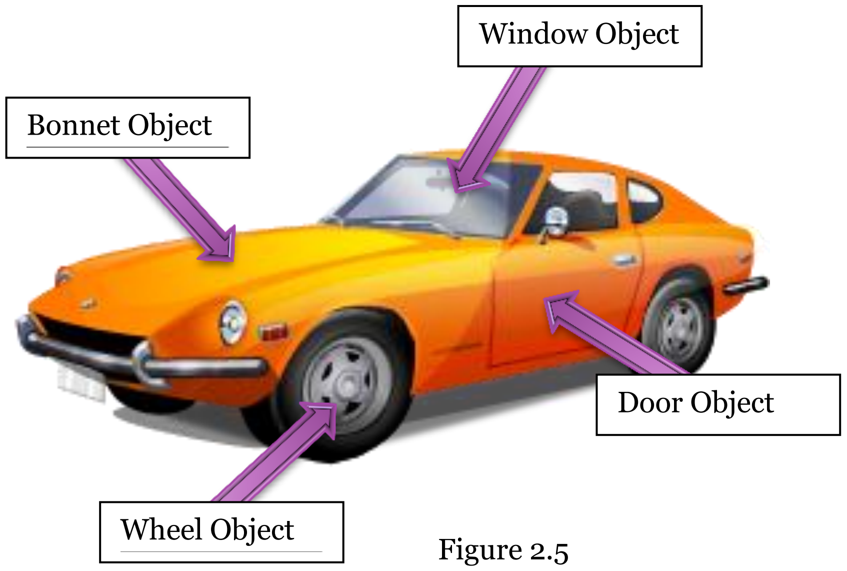


Figure 2.5

Cars are objects. Cars can contain other objects like a Wheel object and a Bonnet object. These objects can contain other objects themselves such as a Tire object for the Wheel object.

VBA works much the same way. A Form object can contain a Command Button object or a Combo-Box object or a Text-box object. And just as a Form is part of a collection, so too can controls be part of a collection.

Taking the car analogy a little further we can make some observations about the car in Figure 2.5:

- The Car Object contains a Door Object.
- The Door object is part of a collection (Doors).
- The count property of the Door object is 2 (There are 2 doors).
- The Door object has a Color property.
- The Color Property of the Door object is set to Orange.
- The Door object contains a Window object.
- The Window object has a Broken property.

- The Broken property of any of the Window objects in this Car object is set to False (None of the windows are broken).
- The Car has a Drive Method.

Get the idea? Objects are things that have attributes (properties) and things they can do (Methods).

Programming with Objects

Parent and Children Objects

In Figure 2.6 we have created an object of type form and saved it (this form is in the downloadable Access content).. We have added three control objects to the form (2 buttons and a text-box), changed some properties, and saved the form with its new objects. The control objects are *children* of the form, whilst the form is the *parent* of the control objects.

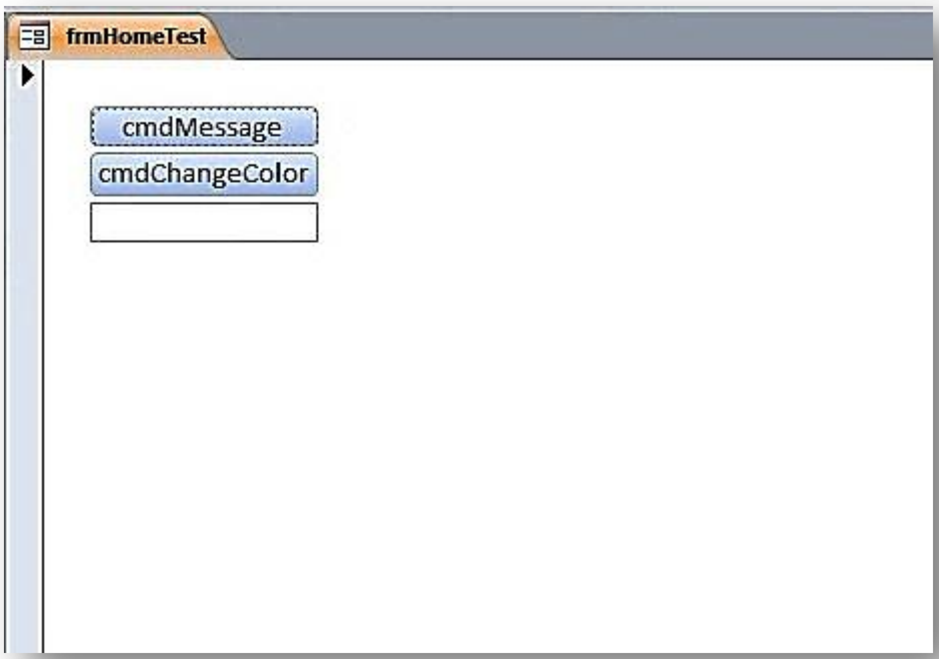


Figure 2.6

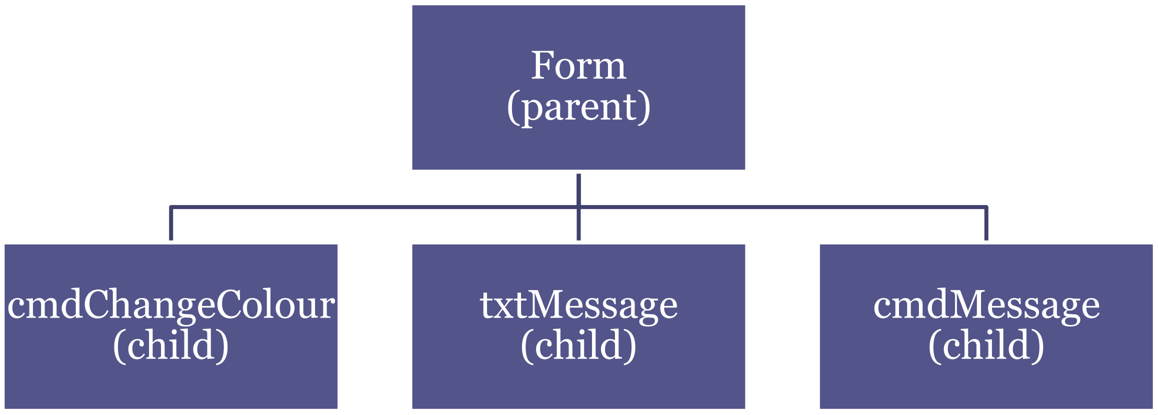


Figure 2.7

Accessing Objects and Properties, Me, the . (dot) Operator and !(exclamation)

We are going to get a message box to appear displaying the text “Hello World!” after clicking on the cmdMessage button. Here’s how we do it:

- Right-Click on frmHomeTest and choose design view from the drop-down box.

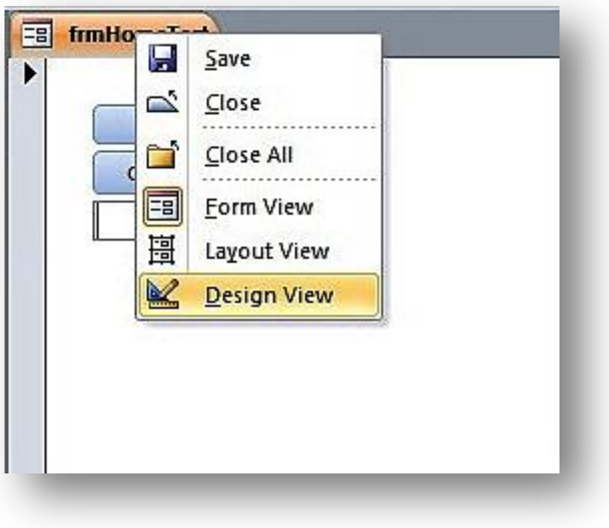
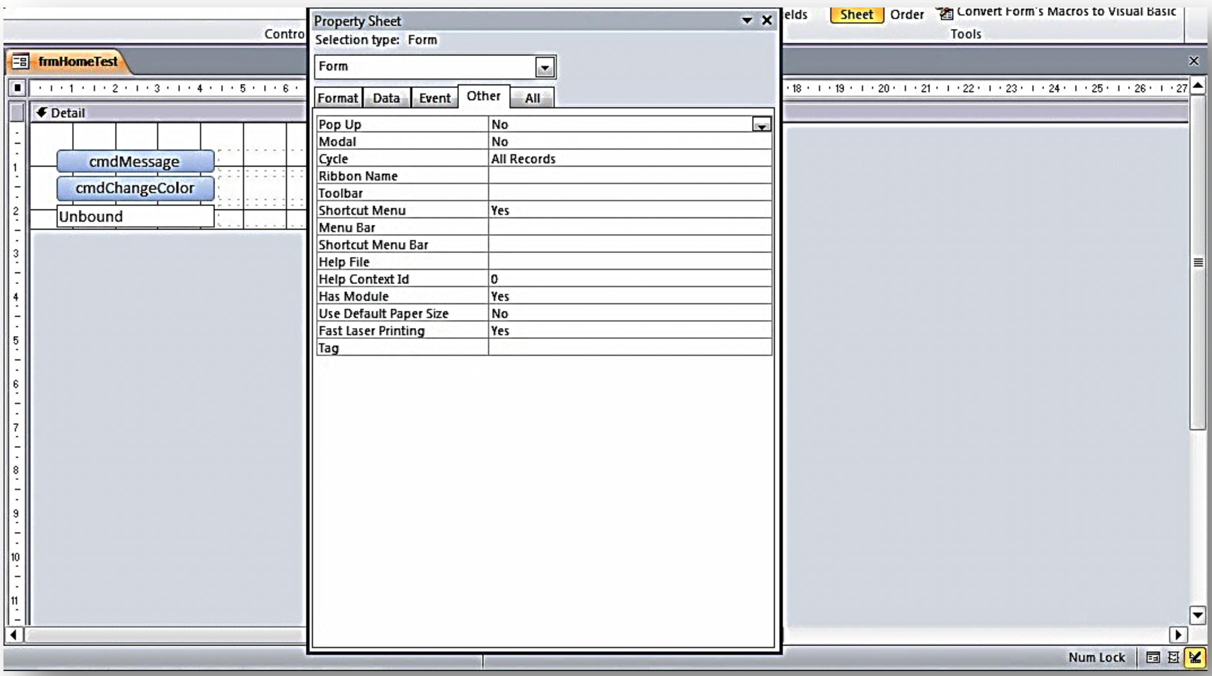


Figure 2.8

- Display the Property Sheet (Press F4).



Property Sheet
Selection type: Form

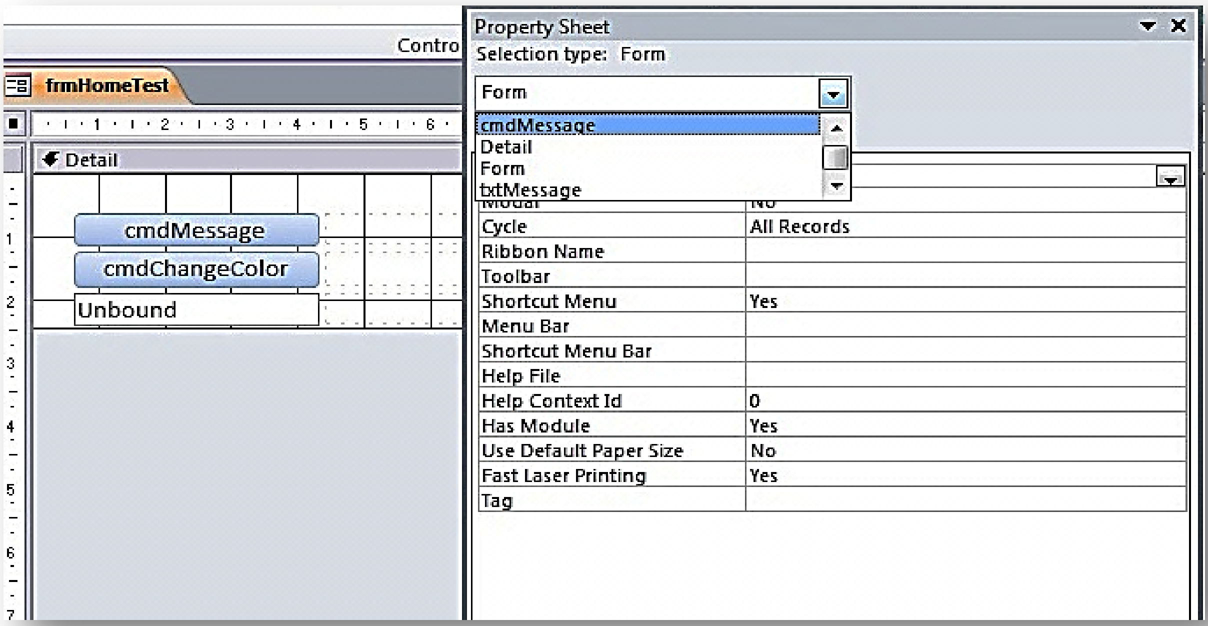
Form

Format | Data | Event | Other | All

Pop Up	No
Modal	No
Cycle	All Records
Ribbon Name	
Toolbar	
Shortcut Menu	Yes
Menu Bar	
Shortcut Menu Bar	
Help File	
Help Context Id	0
Has Module	Yes
Use Default Paper Size	No
Fast Laser Printing	Yes
Tag	

Figure 2.9

- Click on cmdMessage in the combo-box.



Property Sheet
Selection type: Form

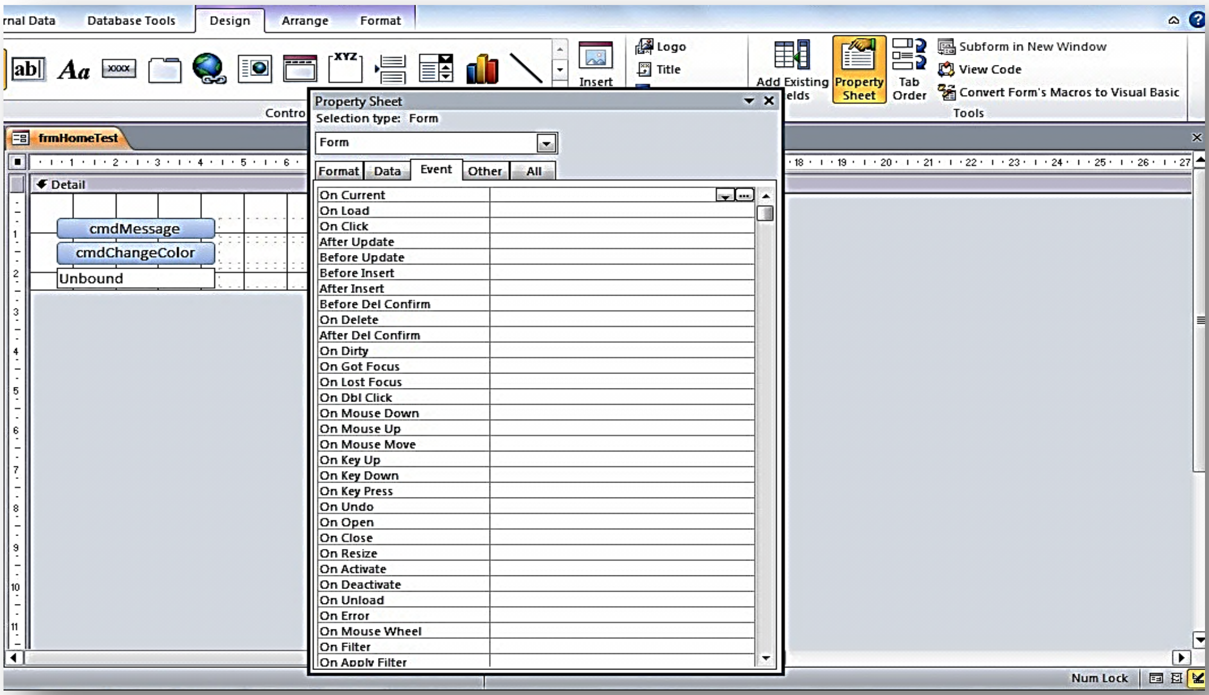
Form

cmdMessage
Detail
Form
txtMessage

Cycle	All Records
Ribbon Name	
Toolbar	
Shortcut Menu	Yes
Menu Bar	
Shortcut Menu Bar	
Help File	
Help Context Id	0
Has Module	Yes
Use Default Paper Size	No
Fast Laser Printing	Yes
Tag	

Figure 2.10

- Go to the Event Tab on the Property Sheet.



Property Sheet

Selection type: Form

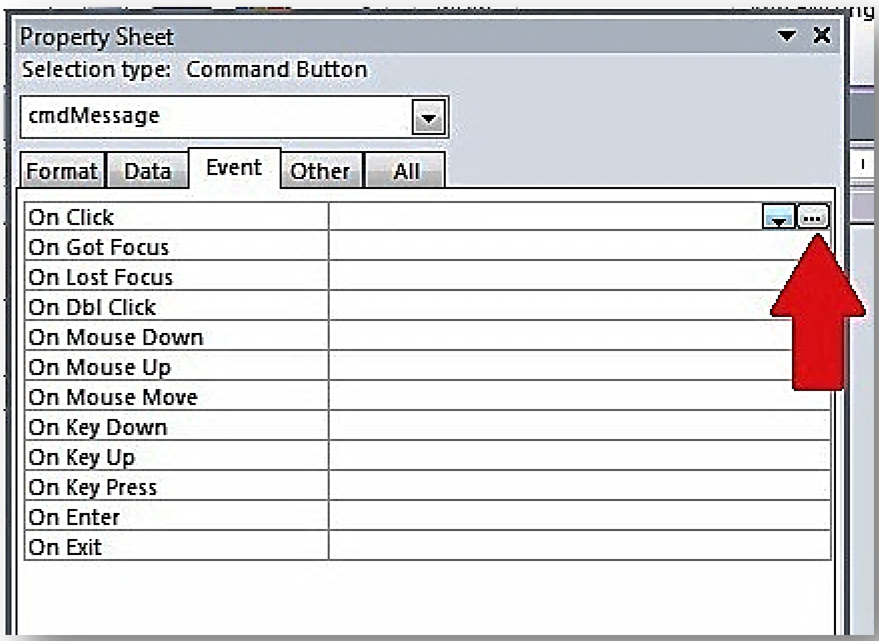
Form

Format | Data | Event | Other | All

On Current
On Load
On Click
After Update
Before Update
After Insert
Before Insert
Before Del Confirm
On Delete
After Del Confirm
On Dirty
On Got Focus
On Lost Focus
On Dbl Click
On Mouse Down
On Mouse Up
On Mouse Move
On Key Up
On Key Down
On Key Press
On Undo
On Open
On Close
On Resize
On Activate
On Deactivate
On Unload
On Error
On Mouse Wheel
On Filter
On Apply Filter

Figure 2.11

- Click on the ellipsis (the 3 dots on the far right) of the On Click Event.



Property Sheet

Selection type: Command Button

cmdMessage

Format | Data | Event | Other | All

On Click
On Got Focus
On Lost Focus
On Dbl Click
On Mouse Down
On Mouse Up
On Mouse Move
On Key Down
On Key Up
On Key Press
On Enter
On Exit

Figure 2.12

- Choose Code Builder.

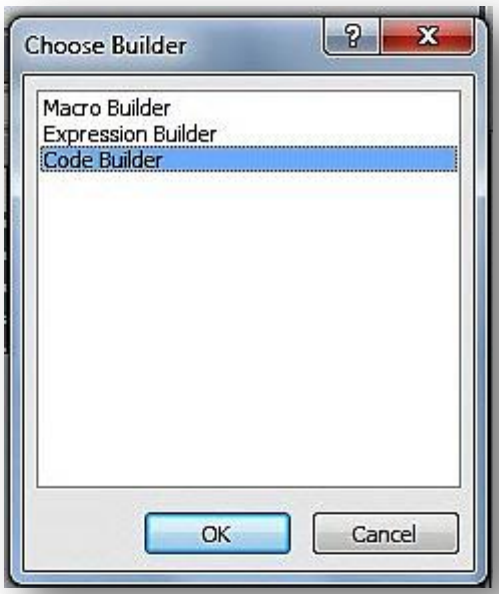
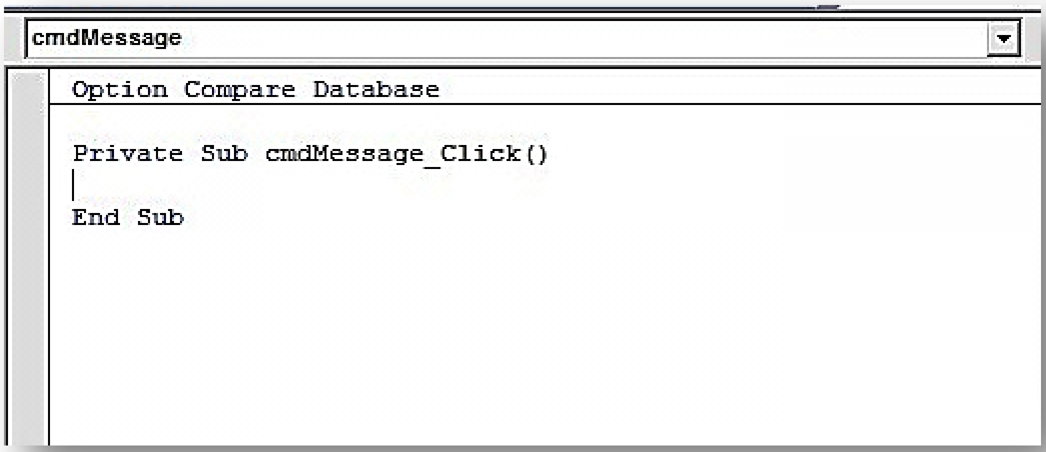


Figure 2.13

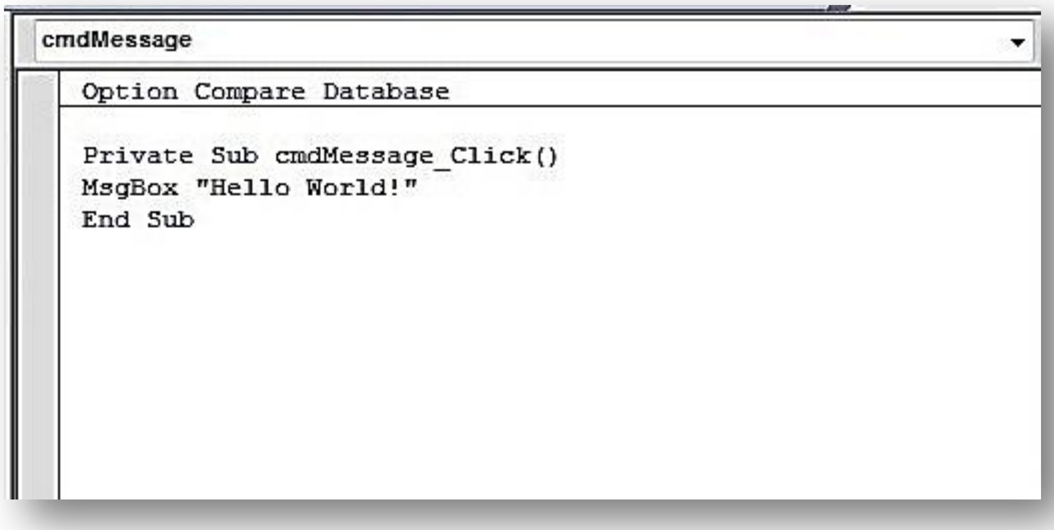
- The shell of a sub-procedure called cmdMessage_Click() will have appeared.



```
Option Compare Database

Private Sub cmdMessage_Click()

End Sub
```

Figure 2.14

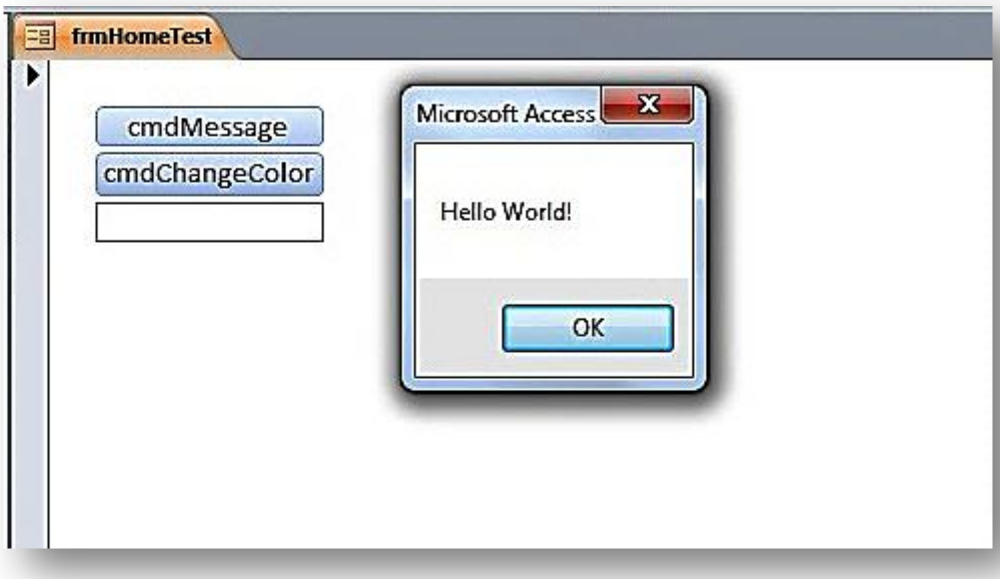
- Write *Msgbox "Hello World!"* on the line below cmdMessage_Click().



```
Option Compare Database

Private Sub cmdMessage_Click()
MsgBox "Hello World!"
End Sub
```

Figure 2.15

If we go back to frmHomeTest and change it Form View we can then click on cmdMessage and the following picture will be displayed:



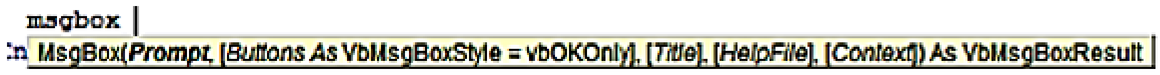
Hello World!

Figure 2.16

We have created an OnClick event for the cmdMessage button and when we press it a dialog box appears that contains the text string "Hello World!". This text string has been **hard-coded**. This means that in order to change the value "Hello World!" we need to alter the code that was written. Sometimes hard-coding is a good thing (as end users don't have access to the code and cannot change certain things) but other times you will want the end users to be able to change certain values. We are going to show how you can get the dialog box to display whatever you write in the txtMessage text box and how these objects can be referenced. But first a quick explanation of how the Msgbox object works.

When you write "MsgBox" and press the space bar once, a yellow box appears listing all the arguments that you can use. This "guide" that VBA provides you with is called intellisence and is a very useful feature.

Note
We will be covering arguments in greater detail later on in the course. For now, think of arguments as things that VBA needs to know to perform certain actions or would like to know (if they are optional). With regards to the msgbox object, VBA needs the Prompt argument (the actual message) otherwise it won't have anything to display!



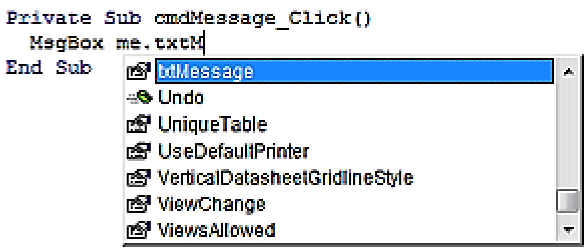
```
msgbox |
MsgBox(Prompt, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult
```

Figure 2.17

- MsgBox(- the open bracket indicates that this object takes arguments.
- *Prompt* is a text String to be displayed.
- *[Buttons As VbMsgBoxStyle = vbOkOnly]* – We use this optional argument to let VBA know what kinds of buttons we want. The default is the OK button.
- *[Title], [HelpFile]* and *[Context]* As *VbMsgBoxResult* – are [optional] and are Strings which we will not concern ourselves with for now.

So the msgbox object will display a text string as the message. In our case we have added a string called "Hello World!" But we don't have to provide the MsgBox object with a *literal* string. We can *reference* an object or variable that contains a string and use that. In other words we can write something in txtMessage and get the msgbox object to display that. Here's how we do it:

- Delete the line *MsgBox "Hello World!"*
- Write *MsgBox* and the press the space bar.
- Write *me.* (not in quotation marks – make sure to include the dot).
- When the yellow box appears write *txt.*
- You should immediately jump to *txtMessage* (as indicated below).
- Choose *txtMessage.*
- Write *.Value* (don't forget the dot).



```
Private Sub cmdMessage_Click()
  MsgBox me.txtM
End Sub
```

Figure 2.18

A couple of points on Figure 2.18.

- "me" – Refers to frmHome1. We are actually writing inside frmHome1 which is why it is highlighted in the Project Explorer.
- . (dot) – to get the controls we've added to the form. We can use the . (dot) operator to bring up a list of Properties of the "me" object. When we dragged the command buttons and textbox onto the form they were added to the Form's list of Properties, and we set the Property's name to txtMessage by changing its name in the Properties window.

Now to test it.

Navigate back to the Form Designer, change the mode to Form View.

Enter "Hello World Again!" into the textbox and click on txtMessage.

There it is!

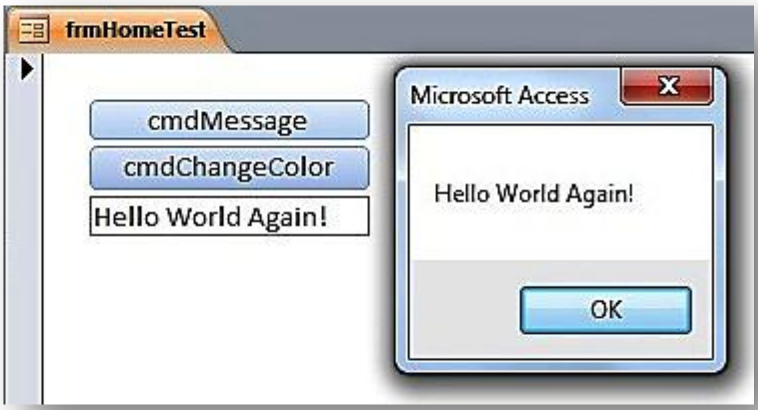


Figure 2.19

Change the text in txtMessage and that is what will appear in the dialog box everytime you press cmdMessage.

Note
Although you can use the dot operator as in Me.txtMessage.Value strictly speaking this should be Me!txtMessage.Value (notice the use of the explanation mark instead of the dot operator). It is good practice to refer to objects with the exclamation mark and properties and methods with the dot operator.

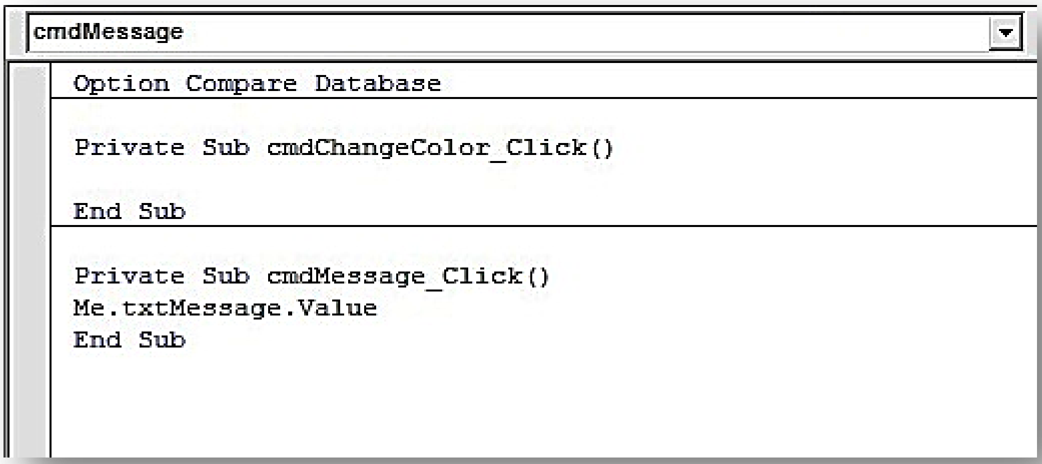
However, if you don't do that the world *will* keep on spinning.

Changing an Object's Properties - textbox background colour

To demonstrate the changing / mutating of an object's properties we are going to do two extra tasks.

1. Change the text on the other button.
2. Change the colour of the textbox background colour.

- Navigate back to the frmHome1 and change it to design view.
- Click on the other command button we called cmdChangeColour.
- On the Property Sheet click events and click the ellipses of the On Click event.
- You should have something that looks like this:

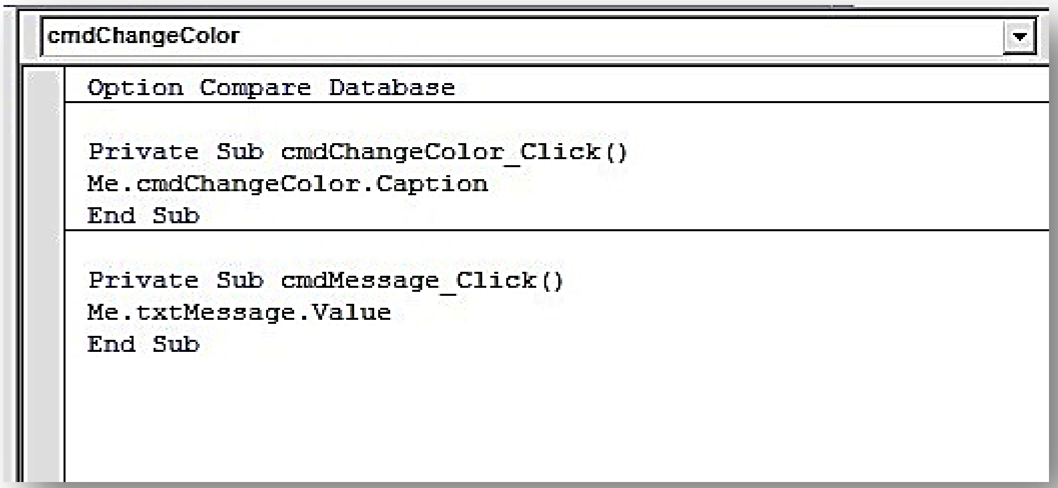


```
Option Compare Database

Private Sub cmdChangeColor_Click()

End Sub

Private Sub cmdMessage_Click()
Me.txtMessage.Value
End Sub
```

Figure 2.20

In the cmdChangeColour_Click() sub do the following:

- Type Me.cmd and find cmdChangeColour
- Put the .(dot) operator
- And find the property Caption

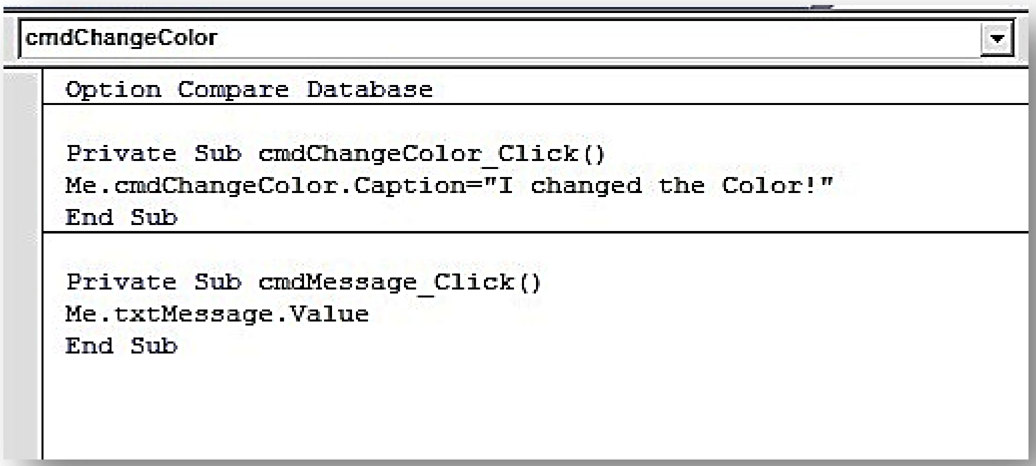


```
Option Compare Database

Private Sub cmdChangeColor_Click()
Me.cmdChangeColor.Caption
End Sub

Private Sub cmdMessage_Click()
Me.txtMessage.Value
End Sub
```

Figure 2.21

Caption is the text that appears in the command button and it's this property we want to set.

- After caption put an "=" sign followed by the string "I Changed the Color!"

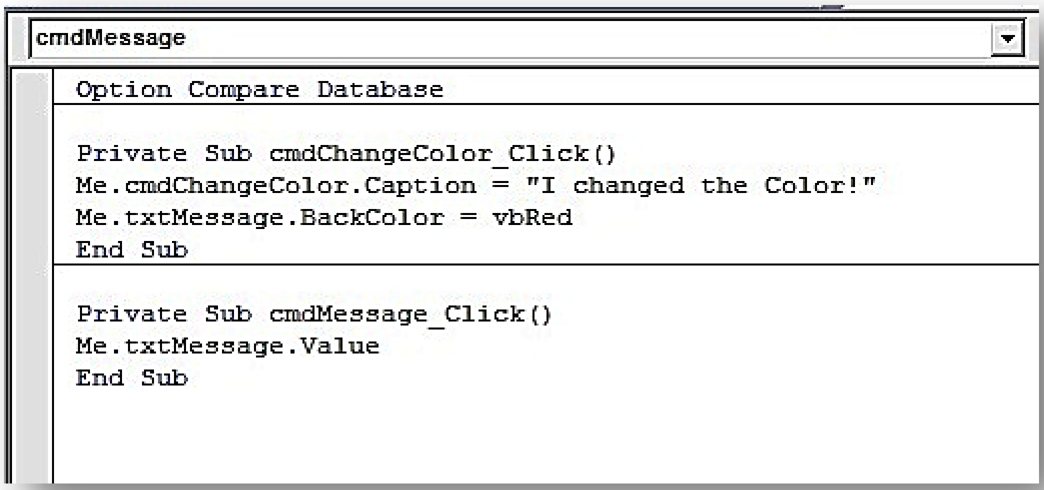


```
Option Compare Database

Private Sub cmdChangeColor_Click()
Me.cmdChangeColor.Caption="I changed the Color!"
End Sub

Private Sub cmdMessage_Click()
Me.txtMessage.Value
End Sub
```

Figure 2.22

On the next line then we will update the background colour of the textbox object.

- Write *Me.txtMessage.Value = VbRed*



cmdMessage

```
Option Compare Database

Private Sub cmdChangeColor_Click()
Me.cmdChangeColor.Caption = "I changed the Color!"
Me.txtMessage.BackColor = vbRed
End Sub

Private Sub cmdMessage_Click()
Me.txtMessage.Value
End Sub
```

Figure 2.23

Now go back to the form in Form View, click the button and it should look like this:

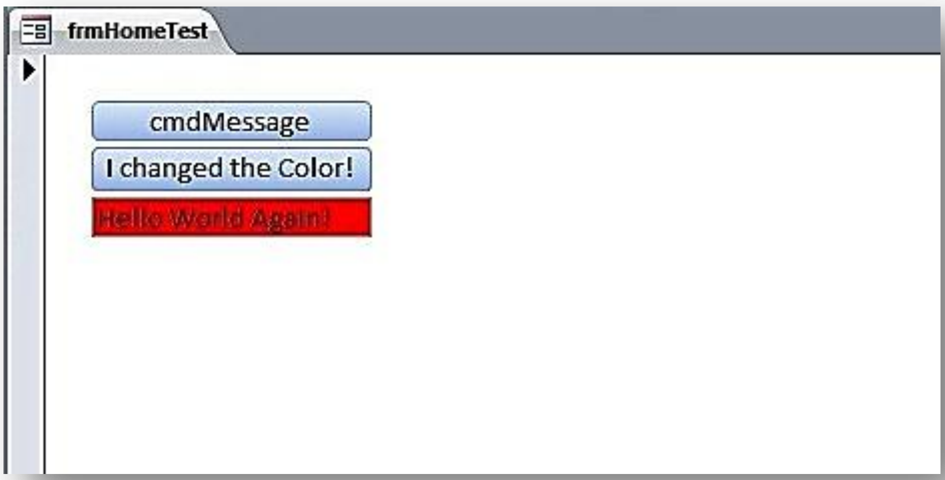


Figure 2.24

Questions

1. Which of these are objects and which are properties:
    a. Command Button
    b. BackColor
    c. Form
    d. Combo-Box
    e. Left
    f. Caption
2. Under what tab is the property sheet on the Ribbon?
3. What shortcut key do you press to bring up the property sheet?
4. Which property determines the color of the text in a command button?
5. Is Movenext a property or method of the recordset object?
6. What does the Count property of a collection tell us?
7. Using the car analogy for objects, properties and methods what category would the color of a tire fit into? Object or property. What object would own that color of a tire.
8. Create a blank form and name it frmObjects. Create a button for the form and name it cmdObjects. What is the relationship between frmObjects and cmdObjects?
9. What property do you change to display the text "This is a button" in the button?
10. How do you create a sub called *Private Sub cmdObjects_Click()*?
11. Add a text box and name it txtObjects. What code do you need to write in Private Sub cmdObjects_Click() to change the Back Color of the text box to vbBlue?
12. When can you use an exclamation mark after the keyword *Me*?

Answers – The VBA Editor, Converting Macros

1. MsgBox doesn't request a user's input
   The line continuation characters are next to the commas, no whitespace
2. a) After Update, b) mouse down, c) detail, d) Detail,
   e) how the form reacts when one is in the last field and tabs once more
3. True or false
   a. False
   b. False
   c. True
   d. False
   e. False
   f. True
   g. False
   h. True
4. a) right click on module in project windows and rename
   b) f2 the module in the navigation panel
5. (b)
6. A camel
7. Multiple answers
   a. Textbox
   b. Integer
   c. Combobox
   d. Label
   e. Long
   f. String
   g. Form
   h. Query
   i. Macro
   j. Class
8. Click Convert Form's Macros to Visual Basic
9. True or false
   a. True
   b. True
   c. False
   d. False
   e. True
   f. False
   g. True
   h. True
   i. True
   j. False
   k. False
10. (b) and (c)
11. **Module 1** and **Module 2** or a similar consecutive numeration
12.
   a. Print is not a function
   b. +_, _ should not be there
   c. Dim is not allowed in the immediate window

    d. Ab(2) is out of bounds, valid values would be 0 or 1

13. Because a form has no module yet.
14. Because the form is open. Forms must be closed to rename.
15. (c)
16. Full Module View.
17. Because they are embedded into a form or report, their parent is the form.
18. Because they are children of their parent form.
19. Public
20. False

Answers – Objects, Properties and Methods

1. The objects are: a,c and d. The properties are b, e and f.
2. The Design Tab.
3. F4.
4. ForeColor.
5. Method.
6. The number of objects within that collection.
7. The color of a tire would be the Color property of the Tire object.
8. frmObjects is the parent object and cmdObjects is the Child Object.
9. The Caption property.
10. The steps to create *Private Sub cmdObjects_Click()*
    a. Make sure the form is in design view.
    b. Bring up the property sheet.
    c. Click on cmdObjects or choose cmdObjects from the drop-down box in the property sheet.
    d. Go to the event tab.
    e. Click on the ellipsis on the far right of the On Click event.
    f. Choose Code Builder.
11. Me.txtObjects.BackColor = vbBlue or Me!txtObjects.BackColor = vbBlue
12. When you wish to refer to an object that is the child of the form or report *me* refers to. E.g. Me!txtObjects (txtObjects is an object).