4

Session Four: An Introduction To VBA

> When I was a kid in school, I learned to read and write
> And every morning in the hall, we sang with all our might
>
> School days are happy days but then you have to go
> And it only takes two hours to put down all you need to know.
>
> ***Children's Song by Neil Innes***

Visual Basic is the most widely used language in the world for building business applications.

Neil Innes assertion works fine with Visual Basic. It really does only take two hours to learn all you need to know to write solid, professional code.

We used to teach the entire VBA language in one session but after years of running our VBA courses in both Excel and Access flavours, decided that this was too ambitious.

This session will get you started in VBA. The code you'll write won't be professional grade yet but you'll be well versed in the basics and be able to write some elementary code. We'll also introduce the debug tools in this session. It may seem odd to learn debug tools before you've even understood the language but you'll find them very useful as you learn more advanced language elements.

The session that follows this one ("Professional Grade VBA") will build upon the basic skills learned in this session and add some vital techniques to make your code professional and robust. This will allow you to begin to do some really useful things with VBA.

Session Objectives

By the end of this session you will be able to:

- Understand subs
- Step through code
- Understand step over, step into and step out
- Understand variables
- Use the immediate window to view and change variable contents
- Use the locals window to view and change variable contents

Session4

© 2007 The Smart Method Ltd

Lesson 4-1: Understand subs

note

Sometimes you'll hear the term Sub Routine used instead of Sub Procedure. You'll also hear some people refer to Sub Procedures as simply Procedures.

None of this terminology is incorrect; we just have many different words that refer to exactly the same thing.

We will simply use the word *Sub* in this book.

Most real-world tasks can be broken down into smaller tasks. In programming we call tasks *procedures*. Procedures can be split into Sub procedures. Consider the procedure of opening your front door. It can be broken down into several sub-procedures:

- Insert key
- Turn key anti-clockwise
- Push door inwards
- Remove key
- Close door

Defining procedures in VBA code

In VBA we cannot leave spaces in procedure names so we would name the above sub-procedures:

InsertKey, TurnKeyAntiClockwise, PushDoorInwards, RemoveKey and CloseDoor.

In VBA code we use the keywords *Sub* and *End Sub* to de-lineate procedures and sub-procedures. There's no distinct difference between a Sub-procedure and a Procedure as they are all de-lineated by the Sub... End Sub statements.

Most VBA programmers refer to sub-procedures simply as *Subs*. The InsertKey sub would thus be typed into the editor as follows:

```
Sub InsertKey()

' InsertKey VBA code will go here

End Sub
```

Note two more things about the above code. We have added parenthesis after the Sub Procedure's name and also a single quotation mark in front of the comment *InsertKey VBA code will go here*. Comments in code are ignored by VBA and can be used to add comments to describe what your code does and how it works.

tip

Always use the Upper/Lower Case naming convention for Sub names (as used in TurnKeyAntiClockwise)

Never use underscores in your own sub names and then it will always be clear which subs are event handlers.

Never, ever, abbreviate words, for example do not use sub names such as TrnKeyAntiClkwse.

These rules (and the reasons for them) are listed in *Appendix A: The Rules*.

Calling procedures in VBA code

To execute all of the sub procedures we have discussed and open the door we would call the subs in sequence like this:

```
Sub OpenDoor()

Call InsertKey
Call TurnKeyAntiClockwise
Call PushDoorInwards
Call RemoveKey
Call CloseDoor

End Sub
```

note

The MsgBox() call is actually to a special type of sub called a function.

Functions can be regarded as being the same as a sub except that they are able to return a value. The MsgBox sometimes returns a value to indicate which message box button was pressed (some MsgBox's have a Yes and No button).

We'll learn all about functions in a later lesson.

tip

You do not have to use the Call keyword when calling a sub or function but most professional programmers would always use this coding style as it makes code far more readable by allowing arguments (if any) to be contained by parenthesis.

For example:-

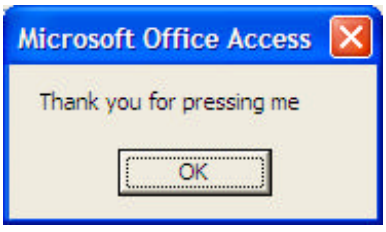
CallMsgBox("Hello")

Instead of:-

MsgBox "Hello"

If you are programming in a team environment it is useful to agree common coding standards such as this so that each member of the team is able to immediately understand other team member's code.

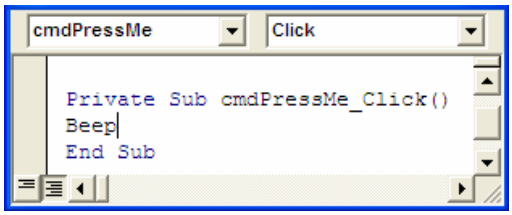
This rule (and others) is listed in *Appendix A: The Rules*.



1 Open the VBACode.mdb database created in the last chapter (if not already open).

2 Open the frmTest form's event handler for the cmdPressMe button's Click event.

Open frmTest in Design View. Right click on the Command Button and select Build Event from the shortcut menu. The code for the Command Button's Click event is displayed in the code editor.



3 Add a new Sub procedure called ThankYouMessage.

Do this by adding the following code:

```
Sub ThankYouMessage()
End Sub
```

4 Remove the Beep statement from the cmdPressMe_Click Sub procedure and add the following code to the ThankYouMessage() Sub procedure:

```
Sub ThankYouMessage()
Beep
Call MsgBox("Thank you for pressing me")
End Sub
```

In its present form the code will not do a thing when the command button is pressed. We need to call the ThankYouMessage sub procedure from the cmdPressMe_Click() event handler.

5 Add a call to the ThankYouMessage sub procedure to the cmdPressMe_Click event handler.

```
Private Sub cmdPressMe_Click()
Call ThankYouMessage
End Sub
```

6 Test the command button.

Display the form in Form View. When you click the command button you should now hear a beep and see the message.

Full code listing

```
Private Sub cmdPressMe_Click()
Call ThankYouMessage
End Sub

Sub ThankYouMessage()
Beep
Call MsgBox("Thank you for pressing me")
End Sub
```

© 2007 The Smart Method Ltd

Lesson 4-2: Step through code

note

The reason that errors in computer code are called bugs is often said to date from an incident at Harvard University in 1945.

There was a problem with a very early mainframe computer when a moth became trapped in the circuits. From then on, when anything went wrong with it the programmers would blame the *bugs*.

While this story is undoubtedly true the reality is that the first recorded use of the word in this context is in the following quote from the *Pall Mall Gazette* of 11 March 1889:

"Mr. Edison, I was informed, had been up the two previous nights discovering a 'bug' in his phonograph - an expression for solving a difficulty, and implying that some imaginary insect has secreted itself inside and is causing all the trouble".

Whenever we write code we're likely to make mistakes. Programmers call mistakes *bugs* for interesting historical reasons (see sidebar).

Now is a great time to introduce one of the VBA Debug tools. The debug tools help you to track down the source of any problems that are encountered when you test your code.

There are many debug tools and we'll be learning about all of them in future lessons. This lesson introduces one of the most useful tools: the ability to step through code.

1 Open the VBACode.mdb database created in the last chapter (if not already open).

2 Open the event handler for the cmdPressMe button's Click event.

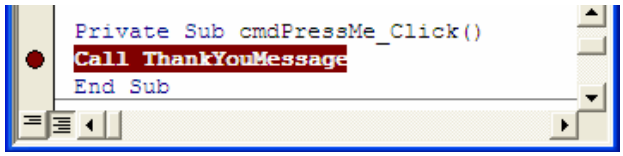
The Code Editor window is displayed.

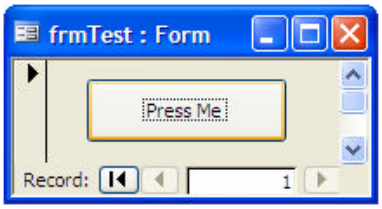
3 Enable the Debug toolbar if it is not already displayed.

You can do this by selecting View➪Toolbars➪Debug from the main menu.

4 Add a breakpoint on the line *Call ThankYouMessage*.

Click anywhere in the text *Call ThankYouMessage* and then click the Toggle Breakpoint button on the Debug toolbar to insert a breakpoint. The selected text is highlighted.





Try clicking the Toggle Breakpoint button twice more to remove and add the breakpoint.

Try clicking in the margin (where you see the brown circle) and note that this method can also be used to toggle breakpoints.

5 Return to Access by clicking the View Access button on the Code Editor's standard toolbar and then change frmTest from *Design View* to *Form View*.

6 Click the cmdPressMe command button.

You are transferred to the code editor with code execution halted at the breakpoint.

Session4a

```
Private Sub cmdPressMe_Click()
Call ThankYouMessage
End Sub

Private Sub ThankYouMessage()
    Beep
    Call MsgBox("Thank you for pressing me")
End Sub
```

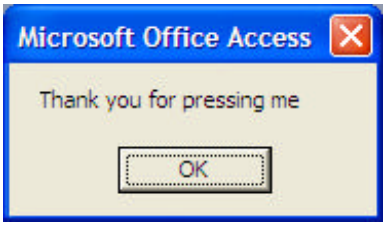
7 Click the Step Into button on the Debug Toolbar twice.

The code steps into the ThankYouMessage sub procedure header and then to the line saying Beep.

```
Private Sub ThankYouMessage()
    Beep
    Call MsgBox("Thank you for pressing me")
End Sub
```

8 Click the Step Into button on the Debug Toolbar once more

An audible Beep is heard as the code executes and the code steps to the next line *CallMsgBox*..




```
Private Sub ThankYouMessage()
    Beep
    Call MsgBox("Thank you for pressing me")
End Sub
```

9 Click the Step Into button on the Debug Toolbar once more

The message box is displayed. Clicking the OK button on the message box steps the code to the next line *End Sub*.

```
Private Sub ThankYouMessage()
    Beep
    Call MsgBox("Thank you for pressing me")
End Sub
```

10 Click the Step Into button on the Debug Toolbar once more

The code moves to the *End Sub* statement from the calling function cmdPressMe_Click.

```
Private Sub cmdPressMe_Click()
Call ThankYouMessage
End Sub
```

11 Click the Step Into button on the Debug Toolbar once more

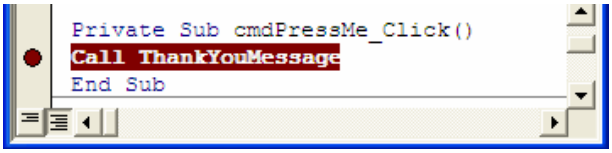
Nothing happens as all code has executed.

12 Remove the breakpoint.

© 2007 The Smart Method Ltd

Lesson 4-3: Understand step over and step out

1 Begin in exactly the same way as the previous lesson by setting a breakpoint at the line *Call ThankYouMessage*.

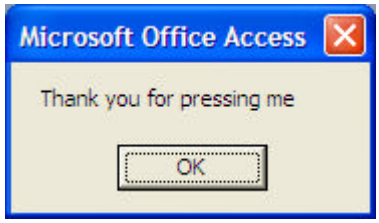


2 Return to the form and Click the cmdPressMe command button.

You are transferred to the code editor with code execution halted at the breakpoint.

3 Click the Step Over button  on the debug toolbar.

An audible Beep is heard as the code in ThankYouMessage executes and the Message Box is immediately displayed.



This happened because the code in the ThankYouMessage sub will execute as normal and will not be stepped through as we requested that VBA *stepped over* the sub routine.

Click the OK button and execution halts at *End Sub* in the cmdPressMe_Click event handler.

This is very useful when you want to examine the way code is executed in a single sub but have no interest in the code within any other subs that it may call.

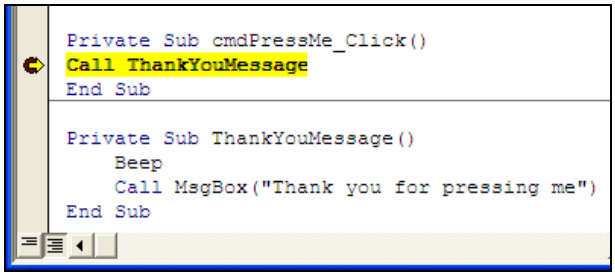
4 Click the Step Over button  (or Step Into button ) on the Debug Toolbar.

Session4a

Nothing happens as all code has executed. In this case *Step Into* and *Step Over* will result in exactly the same action.

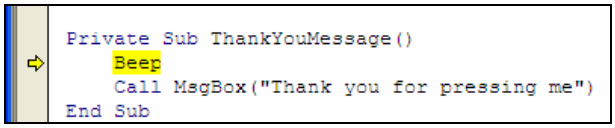
5 Click the cmdPressMe command button.

You are transferred to the coded itor with code execution halted at the breakpoint.



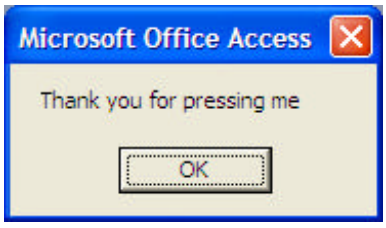
6 Click the Step Into button  on the Debug Toolbar twice.

The code steps into the ThankYouMessage sub procedure header and then to the line saying Beep.



7 Click the Step Out button  on the Debug Toolbar once more.

An audible Beep is heard as the code executes and the message box is immediately displayed.



8 Click the OK button on the displayed message box.

Code execution is now halted at the next line of code in the cmdPressMe_Click event handler. The StepOut button has executed all remaining code in the ThankYouMessage sub procedure without stopping and then paused on the next executable line of the calling sub procedure—cmdPressMe_Click.

StepOut is very useful when you are no longer interested in the remaining code within the current sub but want to examine how the remainder of the code in the calling sub will execute.

© 2007 The Smart Method Ltd

Lesson 4-4: Understand variables

note

Why the strange variable name?

You may have thought that we would call a variable that will store a FirstName simply *FirstName* rather than *strFirstName*.

While VBA wouldn't complain about a variable named *FirstName* we use the prefix *str* in order to show that the variable contains string (text) information rather than, for example, a date or a number.

You'll learn more about the reason why this is a good idea when we cover data types more thoroughly in a later lesson.

Variable naming conventions are all documented in *Appendix A: The Rules*.

What are variables?

A variable is a container that is able to store data of any type in memory.

Every *variable* has a *data type*. For example a variable may be a number, text or a date.

Suppose we have two variables called *strFirstName* and *strLastName*.

Look at the following VBA code:

```
strFirstName = "Freddie"
strLastName = "Mercury"
strFullName = strFirstName & " " & strLastName
```

The strFullName variable now contains the text *Freddie Mercury*. Note the concatenation operator (&) used to append a space (" ") onto the text *Freddie* and the text *Mercury* after the space.

1 If it is not already open, open the VBACode.mdb database.

2 Open the frmTest form in Design View.

3 Open the VBA Editor.

When the form is in Design View you will notice a Code Button  on the top toolbar. Clicking this button will take you straight to the Code Editor and is quicker than right-clicking the command button and selecting Build Event from the shortcut menu.

4 Add the following code to the ThankYouMessage sub procedure:

```
strMessage = "Thank you for pressing me."
```

5 Replace the message text in the MsgBox function call with your new variable.

Because the variable strMessage contains the message text we can now use the variable in place of the text originally entered:

```
Call MsgBox( "Thank you for pressing me" )
```

Becomes

```
Call MsgBox( strMessage)
```

Your code should now look like this:

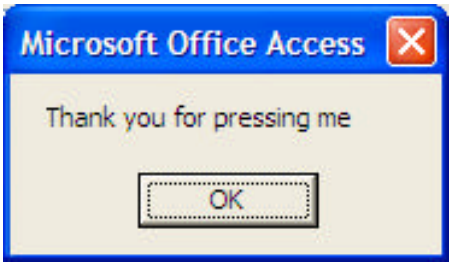
```
Private Sub cmdPressMe_Click()
Call ThankYouMessage
End Sub

Sub ThankYouMessage()
strMessage = "Thank you for pressing me"
Beep
Call MsgBox(strMessage)
End Sub
```

Session4a

6 Test the command button.

Switch to Form view. When you click the command button you should still hear a beep and see the message displayed in a dialog box proving that your new variable is working properly.



© 2007 The Smart Method Ltd

Lesson 4-5: Use the immediate window to view and change variable contents

We've already seen how easy it is to set breakpoints and to step through code using the debug tools.

Programmers often want to examine the contents of different variables at particular points in code execution. The Immediate window lets you do just that. It's hard to exaggerate how useful the Immediate window is when debugging code.

1 Set a break point in your code on the Beep command in the ThankYouMessage sub routine.

2 Click the command button on your frmTest form to begin code execution.

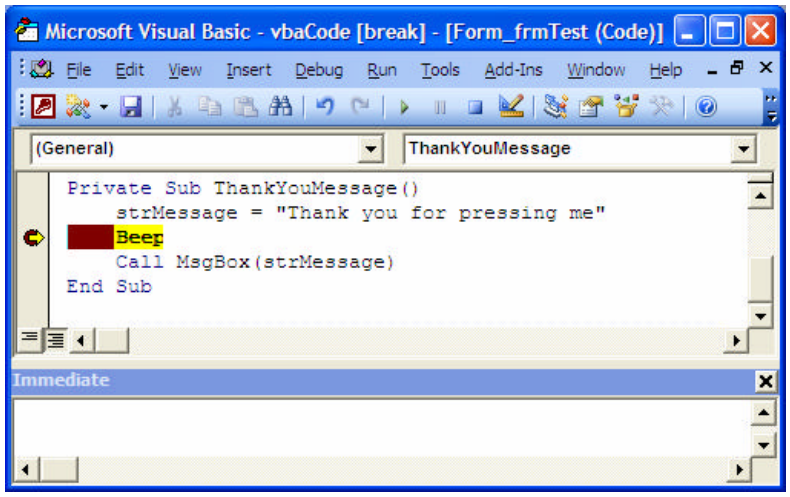
The code editor window opens with code execution halted at the Beep command.

```
Private Sub cmdPressMe_Click()
Call ThankYouMessage
End Sub

Private Sub ThankYouMessage()
    strMessage = "Thank you for pressing me"
    Beep
    Call MsgBox(strMessage)
End Sub
```

3 Enable the Immediate Window if it is not already visible.

The immediate window will be at the bottom of the code window if it is enabled. If not select View ➪ Immediate Window from the main menu to make it appear.

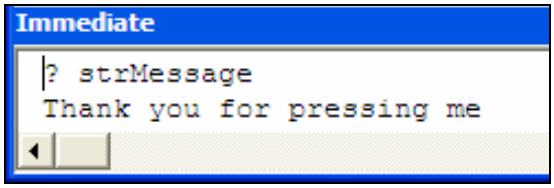


4 Display the contents of the strMessage variable in the immediate window.

Session4b

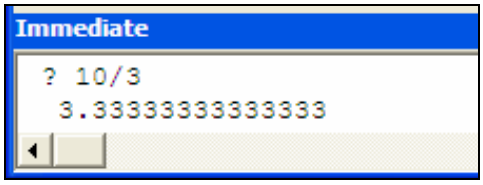
Type the text *? strMessage* in the immediate window and then press the <Enter> key. The question mark is a throw-back to the very first version of the Basic language and means "Print". You will use the question mark a lot in the immediate window.

The contents of the variable strMessage is displayed.



5 Display the literal result of a calculation in the Immediate window.

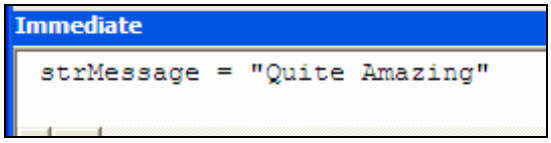
You can type any arithmetic expression into the immediate window. For example type *? 10/3* and then press the <Enter> key. The answer is displayed.



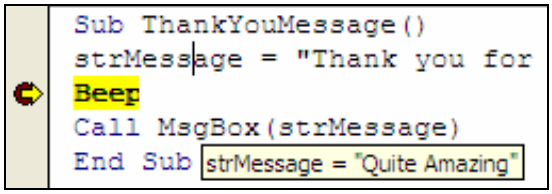
6 Change the value of the strMessage variable using the immediate window.

Sometimes it is useful to change the values of a variable at a break point in your code.

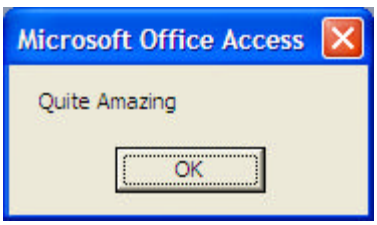
Type *strMessage = "Quite amazing"* in the immediate window followed by the <Enter> key.



7 View the new value of strMessage using an Auto Data Tip.



If you hover the mouse cursor over either instance of strMessage within your code an Auto Data Tip will appear showing the current contents of the variable.



8 Click the Continue button  on the Debug toolbar to execute the remainder of the code.

The message box is displayed with the new message.

© 2007 The Smart Method Ltd

Lesson 4-6: Use the locals window to view and change variable contents

If you thought that the immediate window was amazing you'll find the locals window even more so.

This window allows you to see the value of all currently available variables (variables that are currently available are said to be *in scope*. We'll be learning more about scope in a later lesson).

1 Set a break point in your code on the Beep command in the ThankYouMessage sub routine.

2 Click the command button on your frmTest form to begin code execution.

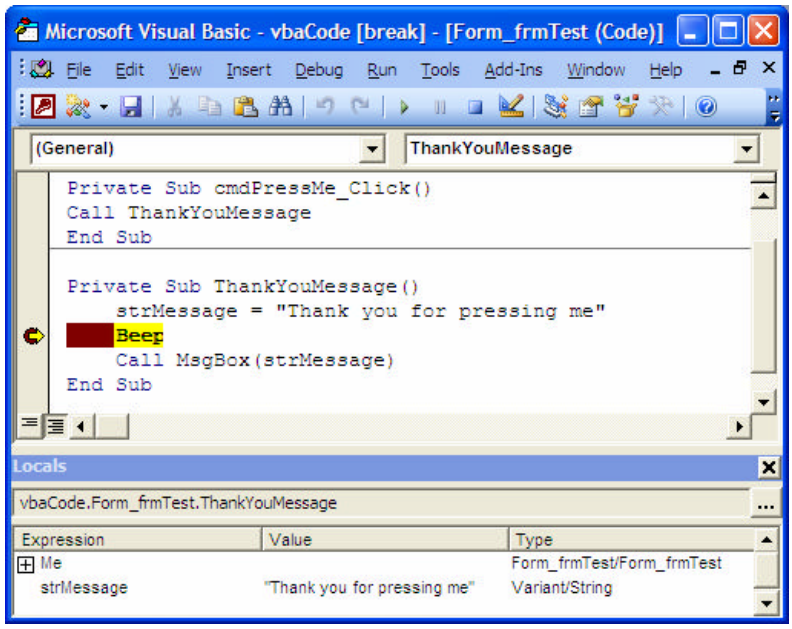
The code editor window opens with code execution halted at the Beep command.

```
Private Sub cmdPressMe_Click()
Call ThankYouMessage
End Sub

Private Sub ThankYouMessage()
    strMessage = "Thank you for pressing me"
    Beep
    Call MsgBox(strMessage)
End Sub
```

3 Click the Locals Window button  on the debug toolbar.

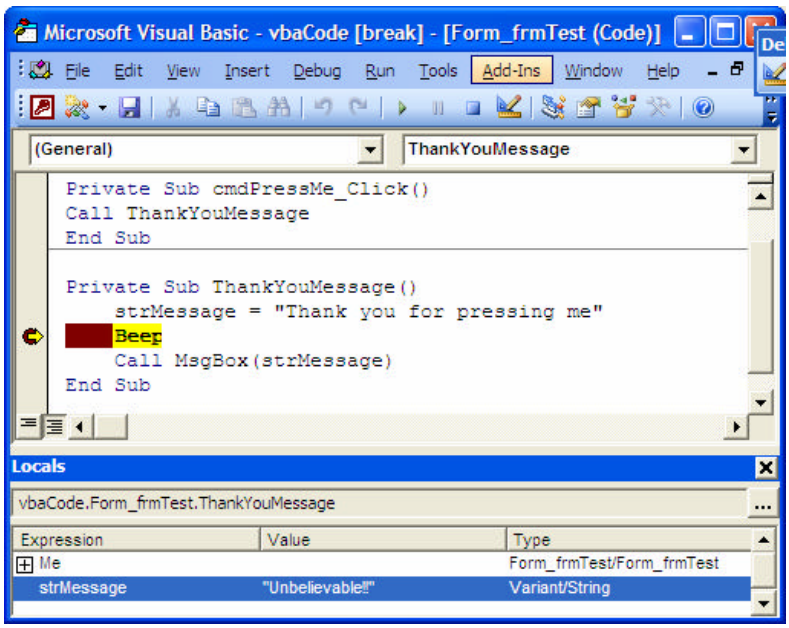
The contents of the variable strMessage is displayed within the locals window.



Session4b

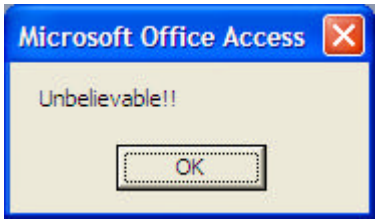
4 Click on the Value "Thank you for pressing me", change the text to "Unbelievable!!" and press the <Enter> key.

The strMessage variable now contains a new value.



5 Click the Continue Button on the debug toolbar.

The message box displays the new contents of the variable strMessage.



The Continue Button is very useful when you are no longer interested in stepping through any more of your code and simply want to resume program execution.

© 2007 The Smart Method Ltd

Session 4: Exercise

1 **Create a new blank database named *Exercise 4*.**

2 **Create a new form and save it with the name *fmTest*.**

3 **Add a command button to the form and set its Caption property to *Log In* with the first letter *(L)* as the hotkey.**

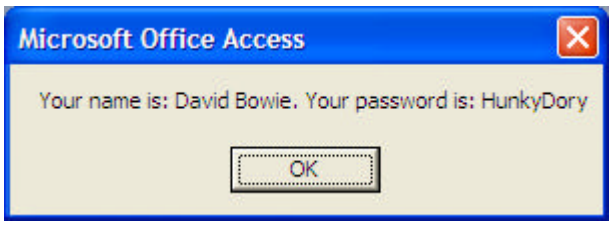
4 **Because the command button has the caption *Log In* there can only be one possible name for the control in order to observe the *Cradle to the Grave Naming Convention*. Set the *Name* property of the command button to the appropriate value.**

When you have finished check that you have correctly set the name and caption of the command button by sliding the page slightly to the left to view the Q4 answer.

5 **Go to the event handler for the command button's Click event. Create two variables there called strLogInName and strLogInPassword containing the following data:**

strLogInName	David Bowie
strPassword	HunkyDory

6 **Add code to the event handler that will put the following dialog on screen by concatenating the text within the two variables.**

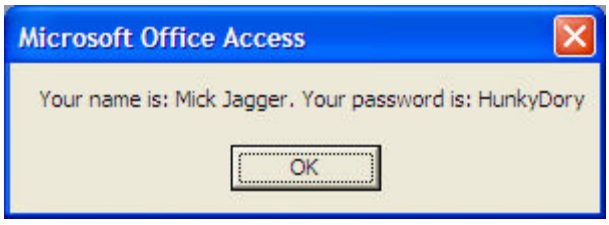


Only if you are unable to do this check the code listing by sliding the page to the left and viewing the Q6 answer.

7 **Set a breakpoint at the *CallMsgBox..* line in your code. Execute your code by clicking the *cmdLogIn* button.**

8 **Use the Immediate window to change the strName value from *David Bowie* to *Mick Jagger*.**

9 **Continue code execution and observe the new message that is displayed.**



If you had any difficulty with this, slide the page to the left for instructions upon how to progress in the Q9 answer.

Exercise4End

© 2007 The Smart Method Ltd

Session 4: Exercise answers

Q 9

To set a break point click in the margin to the left of the code line beginning with **CallMsgBox..**

Return to the form, enter Form View and then click the command button. Code execution will halt at the break point.

Bring up the Immediate window by selecting *View✍Immediate Window* from the main menu.

Type in the Immediate Window:

strLogInName="Mick Jagger" followed by the <Enter> key.

Continue code execution by

clicking the *Continue button*  on the Debug toolbar.

Q 6

```
Private Sub cmdLogIn_Click()
strLogInName = "David Bowie"
strPassword = "HunkyDory"
Call MsgBox("Your name is: " & strLogInName & ". Your password is: " & strPassword)
End Sub
```

Q 4

You should have typed **&Log In** as the caption for the command button.

The command button name should be:

cmdLogIn