Visual Basic for Applications Hands-on Exercises: Track 1

Created: February 1994

Abstract

This document provides an overview of Visual Basic®, Applications Edition, in Microsoft® Excel version 5.0 by way of examples. The following exercises are provided:

- Macro Recording
- Cell Referencing
- With Clauses
- Calling One Macro from Another
- On-Sheet Controls—Drop-Down List Boxes
- Dialog Boxes
- More On-Sheet Controls—Spinners and Radio Buttons
- Message Box, Send Mail, and If Statements
- Variables, For Loops, and Chart Animation
- Visual Basic Toolbar
- Break Points and Watch Variables
- Visual Basic for Applications Help
- OLE Automation—Microsoft Word
- OLE Automation—Microsoft Project
- Important Terms

The files necessary to complete the exercises in this article are available in the HANDSON sample application in the Microsoft Development Library.

Macro Recording

Macro recorders provide a useful aid to end users in creating macros. Macro recorders are also useful for developers in recording complex, unfamiliar sequences. Features of the Microsoft Excel macro recorder include:

- Assigning names to macros.
- Storing descriptions with macros.
- Adding macro commands to the Tools menu.
- Macro recording in either Visual Basic for applications or the Microsoft Excel 4.0 XLM macro language.

Exercises

1. Launch Microsoft Excel. Load file ***NSM1.XLS***.

2. Select ***Tools-Record Macro-Record New Macro.***
3. Type ***Enter_Data*** in the Macro Name box.
4. Enter a short description in the Description box.
5. Click the ***Options*** button.
6. Select ***Visual Basic for Applications*** for the macro language.
7. Select ***OK***.
8. Fill in the table with your first and last name, position, and years at Microsoft.
9. When you are finished, click the ***Stop Record*** button.
10. Select the Module1 tab. The recorded code should appear as follows:

```
'
' Enter_data Macro
' This macro is used to enter data.
'
Sub Enter_data()
    Range("C6").Select
    ActiveCell.FormulaR1C1 = "Joe"
    Range("C7").Select
    ActiveCell.FormulaR1C1 = "Smith"
    Range("C8").Select
    ActiveCell.FormulaR1C1 = "District Manager"
    Range("C9").Select
    ActiveCell.FormulaR1C1 = "5"
    Range("C10").Select
End Sub
```

11. Go back to the Forecast Worksheet by clicking on the Forecast Worksheet tab at the bottom of the screen.
12. Delete the entries you just made in the table in the upper portion of the screen.
13. From the ***Tools*** menu, select ***Macro*** and then ***Enter_data***.

Cell Referencing

Microsoft Excel gives developers several different options for referencing cells in a macro. Microsoft Excel offers both absolute and relative referencing as well.

Exercises

1. Go back to Module1 (by clicking the Module1 tab).
2. Click the right mouse button on the Module1 tab and select ***Delete*** from the shortcut menu to delete the module.
3. Go back to the Forecast Worksheet by clicking the Forecast Worksheet tab. Click the right mouse button on the Forecast Worksheet tab and select ***Insert*** from the shortcut menu. Select ***Module*** and then ***OK***. This will add Module2 to your workbook.
4. Enter the following simple macro:

```
Sub Enter_Data()
```

```
   range("first_name")="Joe"
End Sub
```

5. Notice that all macros start with a Sub statement and end with an End Sub statement.
6. Also notice that whenever you enter keywords, Visual Basic for applications highlights them in blue for you automatically.
7. Go back to the Forecast Worksheet and delete the entries for first name, last name, position, and years at Microsoft. Select ***Tools-Macro-Enter_Data*** to enter Joe in the ***first_name*** box.
8. As you can see, the cells in the table have defined names that can be used to program the cells directly.
9. There are several different ways to reference cells using Visual Basic for applications. Go back to Module2 and edit the macro so that it looks as follows:

```
Sub Enter_data()
    Range("first_name") = "Joe"
    Range("c7") = "Smith"
    Cells(8,3) = "District Manager"
    Worksheets("Forecast Worksheet").Range("years") = 10
End Sub
```

10. Go back to the Forecast Worksheet and run ***Tools-Macro-Enter_data***.

With Clauses

"With" clauses make it very easy to write macros by allowing developers to abbreviate commands.

- "With" clauses allow developers to save time.
- "With" clauses actually reduce the amount of code required to perform a task.

Exercises

1. Enter the following macro:

```
Sub format_data()
    With Range("first_name").Font
        .Name = "times new roman"
        .Bold = True
        .Italic = True
        .size = 12
    End With
End Sub
```

2. Go back to the Forecast Worksheet and run ***Tools-Macro-Format_data***.
3. Go back to Module2. Change the first command in the ***format_data()*** macro to the following:

```
With Range("First_name:years").font
```

4. Go back to Forecast Worksheet and run ***Tools-Macro-Format_data***.

Calling One Macro from Another

Visual Basic for applications macros are modular in that they can be shared with other users or developers. With Microsoft Excel, users can:

- Call one Visual Basic for applications macro from another Visual Basic for applications macro.
- Call a Visual Basic for applications macro in one application (Microsoft Excel) from a Visual Basic for applications macro in another application (Microsoft Project).
- Call an XLM macro from a Visual Basic for applications macro.
- Call a Visual Basic for applications macro from an XLM macro.
- Call a Visual Basic for applications or an XLM macro from a Lotus® 1-2-3® macro.

Exercises

1. Select Range C6:C9 of the Forecast Worksheet. Display the Formatting toolbar (if the formatting toolbar is not displayed, select ***View-Toolbars-Formatting***). Undo the formatting of the selected range by clicking the ***Bold*** button and the ***Italic*** button, and setting the font size to 10. Then press the ***Delete*** key to delete the entries.
2. Go back to Module2. Insert the following command as the last line of the ***Enter_data*** macro:

   ```
   Format_data
   ```

   The entire macro should now look as follows:

   ```
   Sub Enter_data()
       Range("first_name") = "Joe"
       Range("c7") = "Smith"
       Cells(8,3) = "Product Manager"
       Worksheets("Forecast Worksheet").Range("years") = "10"
       format_data
   End Sub
   ```

3. Go back to the Forecast Worksheet. Select ***Tools-Macro-Enter_data***.

On-Sheet Controls--Drop-Down List Boxes

On-sheet controls in Microsoft Excel 5.0 make it very easy to transform a simple worksheet into a powerful form. On-sheet controls in Microsoft Excel 5.0 are available on the Forms toolbar and include:

- Drop-Down List Boxes
- Radio Button Boxes
- Check Boxes
- Sliders
- Spinner Controls

Exercises

1. Display the Forms toolbar by selecting ***View-Toolbars-Forms***. Anchor the Forms toolbar at the top of the screen.
2. Click the drop-down control box tool (the 8th tool from the left). Use Microsoft Excel 5.0 Tool Tips to help you find the right tool.

3. Draw a drop-down control in cells C8:D8.
4. Select the drop-down control and click the right mouse button.
5. Select ***Format Object*** and then the ***Control*** tab.
6. Select the ***Input Range*** box. Then click the worksheet tab labeled List. Select the range A1:A5 on the List worksheet.
7. Select ***OK***.
8. When you return to the Forecast Worksheet, use the drop-down control to select different positions.

Dialog Boxes

Microsoft has made a vast improvement over the way that dialogs have been created in Microsoft Excel in the past:

- Dialog boxes are edited and saved on their own dialog sheets.
- The Forms toolbar provides the same dialog controls that users can place on worksheets.

Exercises

1. Click the Forecast Worksheet tab with the right mouse button.
2. Select ***Insert-Dialog-OK.***
3. For the dialog box title, type "Years at Encore".
4. Select the label tool on the Forms toolbar (1st tool on the left). Draw a label in the upper portion of the dialog and type "Number of years at Encore".
5. Click the drop-down control tool (8th from the left). Draw a drop-down list box on the dialog next to the label.
6. Right-click the drop-down list box and select ***Format Object*** from the Shortcut menu.
7. Select the ***Control*** tab and then select Input Range. Select the List worksheet by clicking on the List workbook tab. Select the range A7:A21 on the List worksheet.
8. Select the Selection Link. Select cell C9 on the Forecast Worksheet.
9. Select 5 for drop-down lines.
10. Select ***OK***.
11. Go to Module2 by clicking the Module2 tab.
12. Enter the following macro at the end of the macro sheet.

```
Sub show_dialog()
    DialogSheets("dialog1").Show
End Sub
```

13. Go to the Forecast Worksheet by clicking the Forecast Worksheet tab.
14. Select the button with the question mark on it in cell D9. Right-click the button, select ***Assign Macro to Object***, and select macro ***show_dialog***. Select ***OK***.
15. Now click the question mark button. Notice how you can use the dialog boxes to make changes to your data.

More On-sheet Controls--Spinners and Radio Buttons

Exercises

1. If the Forms toolbar is not displayed, select ***View-Toolbars-Forms***.
2. Click the spinner control on the Forms toolbar (12th tool from the left). Use Microsoft Excel 5.0 Tool Tips to help you find the right tool.
3. Draw a spinner control in cell C9.
4. Right-click the spinner and select ***Format Object*** from the shortcut menu. Select cell-link and select cell C9 on the Forecast Worksheet. Select ***OK***.
5. Use the spinner control to adjust the number of years.
6. Select the Radio Button control from the Forms toolbar (5th tool from the left). Draw another Radio Button entry in the "Survey: NSM 1994" box under Issaquah.
7. Type "Tahiti" for the caption of the button.
8. Click the various radio buttons in the survey box. Notice how the selection number changes when you select new buttons.

Message Box, Send Mail, and If Statements

The Visual Basic for applications Message Box makes it very easy to display a message to your user.

Integration with MAPI makes it easy to create custom applications with the Microsoft Office that allow users to share data.

Exercises

1. Go to Module2 by clicking the Module2 tab.
2. Enter the following macro:

```
Sub send_survey()
    If (Range("selection") = 4 And Range("growth") < .5) Then
        MsgBox "Sorry. You must have at least 50% growth to go to
Tahiti."
    Else
        ActiveWorkbook.SendMail ("")
    End If
End Sub
```

3. Go back to the Forecast Worksheet by clicking the Forecast Worksheet tab.
4. Right-click the ***Send Survey*** button and select ***Assign Macro to Object***. Select the ***Send_Survey*** macro.
5. Start Microsoft Mail by switching to the Windows™ Program Manager and double-clicking the mail icon.
6. Click the ***Send Survey*** button. Notice that if Tahiti is selected and if growth is below 50%, you receive an error message. However, if either Tahiti is not selected, or Tahiti is selected and growth is above 50%, then the workbook will be packaged in an e-mail message.

Variables, For Loops, and Chart Animation

Exercises

1. Go back to Module2 and enter the following macro:

```
Sub beep_count()
    Dim total_growth As Single
    Dim count1 As Single
    total_growth = Range("growth")
    For count1 = 0 To total_growth Step .05
        Range("growth") = count1
        Beep
    Next
End Sub
```

2. The Dim statement is used to declare a variable—in this case total_years as a "single".
3. The For loop scrolls through the number of years and then beeps for every year.
4. Go to the Forecast Worksheet. Place a button on the worksheet by first displaying the Drawing toolbar (View-Toolbars-Drawing) and then selecting the button tool (13th from the left). Draw a button next to the cell displaying the Annual Growth rate on the Forecast worksheet.
5. You'll be prompted to assign a macro to the button you've just added. Select the macro ***beep_count***. Change the caption of the button to "Count".
6. Click the ***Count*** button.
7. Go to Module2 by clicking the Module2 tab. Change the ***beep_count*** macro so that it now looks like this (statements in bold have been added):

```
Sub beep_count()
    Dim total_growth As Single
    Dim count1 As Single
    total_growth = Range("growth")
    Charts("chart1").Select
    For count1 = 0 To total_growth Step .02
        Range("growth") = count1
        Beep
    Next
    Worksheets("forecast worksheet").Select
End Sub
```

8. Go back to the Forecast Worksheet and click the ***Count*** button again to view chart animation.
9. Go back to Module2 and add the For loop to your macro (highlighted in bold below):

```
Sub beep_count()
    Dim total_growth As Single
    Dim count1 As Single
    total_growth = Range("growth")
    Charts("chart1").Select
    For count1 = 0 To total_growth Step .02
        Range("growth") = count1
        Beep
    Next
    For x = 40 to 220 step 10
      Charts("Chart1").rotation = x
```

```
    Next
    Worksheets("forecast worksheet").Select
End Sub
```

10. Now go back and run the ***beep_count*** macro again—in addition to the chart-growing effect, you now have chart rotation.

Visual Basic Toolbar

The Visual Basic for applications Toolbar gives you instant access to common editing commands:

- Insert Module
- Edit Menus
- Object Browser
- Run, Step, and Resume macros
- Start and Stop recording macros
- Breakpoints
- Watch Variables
- Step Into and Step Over

Exercises

1. Go to Module2 by clicking the Module2 tab.
2. ***Insert Module*** button (on the Visual Basic toolbar)—used to create a new module.
3. Menu Editor (Visual Basic toolbar)—used to create custom menus.
4. Object Browser (Visual Basic toolbar)—used to access Microsoft Excel, Visual Basic for applications, and application objects, properties, and methods.
5. Click the ***Object Browser*** button. Select the ***Excel*** library. Select the ***Worksheet*** object, then select ***Range***. Select ***Cancel***.
6. VCR Panel (Visual Basic toolbar)—Run, Step, Resume, Stop, and Record macros.

Break Points and Watch Variables

Break Points provide a useful tool in debugging code in that they allow you to stop execution at any point in the macro.

Watch Variables are also very useful in debugging macros in that they allow you to see the value of any variables (or formulas) at any time during the execution of a macro.

Exercises

1. Edit the ***beep_count*** macro so that it now looks like this (added statements are in bold):

```
Sub beep_count()
    Dim total_growth As Single
    Dim count1 As Single
    total_growth = Range("growth")
```

```
    Charts("chart1").Select
    For count1 > 0 To total_growth Step .05
        Range("growth") = count1
        Beep
        If count1 = (total_growth-.05) Then
            Beep
        End If
    Next
    For x = 40 to 220 step 10
      Charts("Chart1").rotation = x
    Next
    Worksheets("forecast worksheet").Select
End Sub
```

2. Select the Beep command after the "If Count1=total_growth-.05" statement and then select the ***Break Point*** button.
3. Select count1 and then press the ***Watch*** variable button and select Add.
4. Select total_growth and then press the ***Watch*** variable button and select Add.
5. Press the ***Run*** macro button from the VCR panel. When the debug window appears, press the ***Watch*** button to see the values of the variables. When you are done, close the debug window.
6. Select the Beep command after the "If Count1=total_growth-.05" statement and press the ***Break Point*** button to remove the break point.

Visual Basic for Applications Help

Visual Basic for applications Help is a valuable tool in accessing online information. Features of Visual Basic for applications Help include:

- Definitions of all objects, properties, and methods.
- Example code that can be copied and pasted directly into your Visual Basic for applications module.
- Definitions of all functions, statements, and key words.
- Tutorial information on writing Visual Basic for applications macros.

Exercises

1. Select ***Contents*** from the Help menu and then select ***Programming with Visual Basic***.
2. Select ***Search***. Enter ***Range*** and select ***Range Object*** and then ***Show Topics***. Select ***Range Object*** from the topics.
3. View the various properties of the Range Object by selecting ***Properties***. Then view the various methods by selecting ***Methods***.
4. Select the Font property (select ***Properties***) of the Range Object and then select ***Example*** to see an example of code for programming the Font property.

OLE Automation--Microsoft Word

Support for OLE Automation in Microsoft Excel (controller and object), Microsoft Project (controller and object), and Microsoft Word (object) makes it very easy for developers to take advantage of the

functionality offered across desktop applications in building custom solutions.

Exercises

1. Go to Module2. Press the ***New Module*** button on the Visual Basic toolbar to create a new module (Module3). Enter the following macro in Module3:

```
Dim msword as object
Sub create_report()
    Set msword = CreateObject("word.basic")
    Range("data_table").Copy
    With msword
        .appmaximize
        .filenewdefault
        .editpaste
    End With
End Sub
```

2. Go to the Forecast Worksheet and select the ***Create Report*** button. Right-click the ***Create Report*** button and select ***Assign Macro to Object*** from the shortcut menu. Select macro ***Create_Report*** and select ***OK***.
3. Click the ***Create Report*** button to create a Word document from your Microsoft Excel table.
4. When the macro is finished executing, close Word and return to Microsoft Excel.

OLE Automation--Microsoft Project

In addition to Microsoft Excel, Visual Basic for applications will also be available in Microsoft Project. Microsoft Project will also support OLE Automation as both a controller and object.

Exercises

1. Go to Module3. Press the ***New Module*** button on the Visual Basic toolbar to create a new module (Module4). Enter the following macro in Module3:

```
Dim MS_Project As Object
sub nsm_sched
    worksheets("nsm schedule").select
end sub
```

2. Go back to the Forecast Worksheet. Right-click the ***NSM Schedule*** button and select ***Assign Macro to Object***. Select the ***nsm_sched*** macro and select ***OK***.
3. Click the ***NSM Schedule*** button to go to the NSM Schedule sheet.
4. Go to Module2 (by clicking the tab) and enter the following macro:

```
Sub ProjTalk()

    Dim Task As Object
     Dim Row As Integer
     Shell ("c:\40spec\proj40\winproj.exe")

    Set MS_Project = GetObject("", "MSProject.Project")
    MS_Project.FileOpen "c:\nsm\project1.mpp"
```

```
    Row = 13

    For Each Task In MS_Project.ActiveProject.Tasks
            Worksheets("nsm schedule").Cells(Row, 3) = Task.Name
            Worksheets("nsm schedule").Cells(Row, 2) =
Task.ScheduledStart
                Row = Row + 1
    Next Task
End Sub
```

5. Go to the NSM Schedule worksheet. Place a button on the worksheet by first displaying the Drawing toolbar (View-Toolbars-Drawing) and then selecting the button tool (13th from the left). Draw a button next to the table displaying the Task Schedule.
6. You'll be prompted to assign a macro to the button you've just added. Select the ***ProjTalk*** macro.
7. Change the caption on the button to read "Update".
8. Click the button to update the NSM Task Schedule.
9. Double-click on the Project icon and close Microsoft Project.

Important Terms

OLE Automation: A feature of OLE 2.0 that allows developers to use Visual Basic for applications in one application to control another application.

OLE Automation Controller: An application that supports OLE Automation to the extent that it can send commands to other applications. Microsoft Excel, Microsoft Project, and Visual Basic 3.0 are OLE Automation controllers.

OLE Automation Object: An application that supports OLE Automation to the extent that it can receive commands from other applications. Microsoft Word, Microsoft Excel, and Microsoft Project are OLE Automation objects (note that Microsoft Excel and Microsoft Project are also controllers).

Object: An object is a part of an application that can be programmed. For example, in Microsoft Excel there are over 120 objects, including worksheets, charts, cells, ranges, and so on.

Property: A property is an aspect of an object—objects are programmed by changing their properties. For example, in Microsoft Excel, the chart object has properties of rotation and elevation. You can program a chart object by specifying different values for the chart's rotation and elevation properties.

Method: A method is an action that is performed on an object. For example, in Microsoft Excel, the range object has a copy method—that is, you can copy the contents of the range to the clipboard.

Procedure: A Visual Basic for applications macro in Microsoft Excel or Microsoft Project.

Module: The actual sheet that contains a Visual Basic for applications macro.