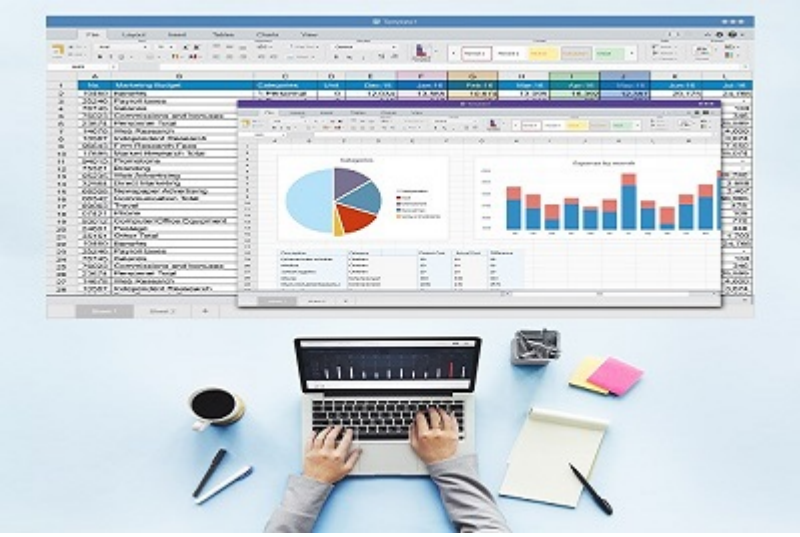
Excel Macro Mastery (https://excelmacromastery.com/)

THE MISSING VBA HANDBOOK

The Complete Guide To The VBA Workbook

DECEMBER 16, 2014 (HTTPS://EXCELMACROMASTERY.COM/EXCEL-VBA-WORKBOOK/) BY PAUL KELLY (HTTPS://EXCELMACROMASTERY.COM/AUTHOR/ADMIN/) · 10 COMMENTS (HTTPS://EXCELMACROMASTERY.COM/EXCEL-VBA-WORKBOOK/#COMMENTS)



"We are drowning in information but starved for knowledge." – John Naisbitt

This post provides a complete guide to using the VBA Workbook.

If you want to use VBA to **Open a Workbook** then check out Open Workbook

If you want to use VBA to **create a new workbook** go to Create New Workbook

For all other VBA Workbook tasks, **check out the quick guide below**.

Contents [hide]

adbox/145db6b73f72a2%3A106f25298346dc/5676073085829120/)

A Quick Guide to the VBA Workbook

The following table provides a quick how-to guide on the main VBA workbook tasks

Task	How to
Access open workbook using name	Workbooks("Example.xlsx")
Access open workbook (the one opened first)	Workbooks(1)
Access open workbook (the one opened last)	Workbooks(Workbooks.Count)
Access the active workbook	ActiveWorkbook
Access workbook containing VBA code	ThisWorkbook
Declare a workbook variable	**Dim** wk **As** Workbook

Task	How to
Assign a workbook variable	**Set** wk = Workbooks("Example.xlsx") **Set** wk = ThisWorkbook **Set** wk = Workbooks(1)
Activate workbook	wk.Activate
Close workbook without saving	wk.Close SaveChanges:=**False**
Close workbook and save	wk.Close SaveChanges:=**True**
Create new workbook	**Set** wk = Workbooks.Add
Open workbook	**Set** wk =Workbooks.Open ("C:\Docs\Example.xlsx")
Open workbook as read only	**Set** wk = Workbooks.Open ("C:\Docs\Example.xlsx", ReadOnly:=True)

Check Workbook exists	**If** Dir("C:\Docs\book1.xlsx") = "" **Then**
	MsgBox "File does not exist."
	EndIf

Check Workbook is open	See Check Workbook Open section below
List all open workbooks	**For Each** wk **In** Application.Workbooks **Debug.Print** wk.FullName **Next** wk
Open workbook with the File Dialog	See File Dialog section below function below
Save workbook	wk.Save
Save workbook copy	wk.SaveCopyAs "C:\Copy.xlsm"
Copy workbook if closed	FileCopy "C:\file1.xlsx","C:\Copy.xlsx"
SaveAs workbook	wk.SaveAs "Backup.xlsx"

Getting Started with the VBA Workbook

We can access any open workbook using the code **Workbooks("Example.xlsm").** Simple replace Example.xlsm with the name of the workbook you wish to use.

The following example shows you how to write to a cell on a worksheet. You will notice we had to specify the workbook, worksheet and range of cells.

```
Public Sub WriteToA1()

' Writes 100 to cell A1 of worksheet "Sheet1" in MyVBA.xlsm
Workbooks("MyVBA.xlsm").Worksheets("Sheet1").Range("A1") = 100

End Sub
```

This example may look a little be confusing to a new user but it is actually quite simple.

The first part up to the decimal point is the Workbook, the second part is the Worksheet and the third is the Range. Here are some more examples of writing to a cell

```
Public Sub WriteToMulti()

' Writes 100 to cell A1 of worksheet "Sheet1" in MyVBA.xlsm
Workbooks("MyVBA.xlsm").Worksheets("Sheet1").Range("A1") = 100

' Writes "John" to cell B1 of worksheet "Sheet1" in MyVBA.xlsm
Workbooks("MyVBA.xlsm").Worksheets("Sheet1").Range("B1") = "John"

' Writes 100 to cell A1 of worksheet "Accounts" in MyVBA.xlsm
Workbooks("MyVBA.xlsm").Worksheets("Accounts").Range("A1") = 100

' Writes the date to cell D3 of worksheet "Sheet2" in Book.xlsc
Workbooks("Book.xlsx").Worksheets("Sheet2").Range("D3") = "1\1\2016"

End Sub
```

You can see the simple pattern here. You can write to any cell in any worksheet from any workbook. It is just a matter of changing the workbook name, worksheet name and the range to suit your needs.

Take a look at the workbook part

```
Workbooks("Example.xlsx")
```

The **Workbooks** keyword refers to a collection of all open workbooks. Supplying the workbook name to the collection gives us access to that workbook. When we have the object we can use it to perform tasks with the workbook.

Troubleshooting the Workbooks Collection

When you use the Workbooks collection to access a workbook, you may get the error message:

Run-time Error 9: Subscript out of Range.

This means that VBA cannot find the workbook you passed as a parameter.

This can happen for the following reasons

1. The workbook is currently closed.
2. You spelled the name wrong.
3. You created e new workbook (e.g. Book1) and tried to access it using Workbooks("Book1.xlsx"). It's name is not **Book1.xlsx** until it is saved for the first time.
4. (Excel 2007/2010 only) If you are running two instances of Excel then Workbooks() only refers to to the workbooks open in the current Excel instance.
5. You passed a number as Index and it is greater than the number of workbooks open e.g. you used Workbooks(3) and only two workbooks are open.

If you cannot resolve the error then use either of the functions in the section Finding all open Workbooks. These will print the names of all open workbooks to the Immediate Window(Ctrl + G).

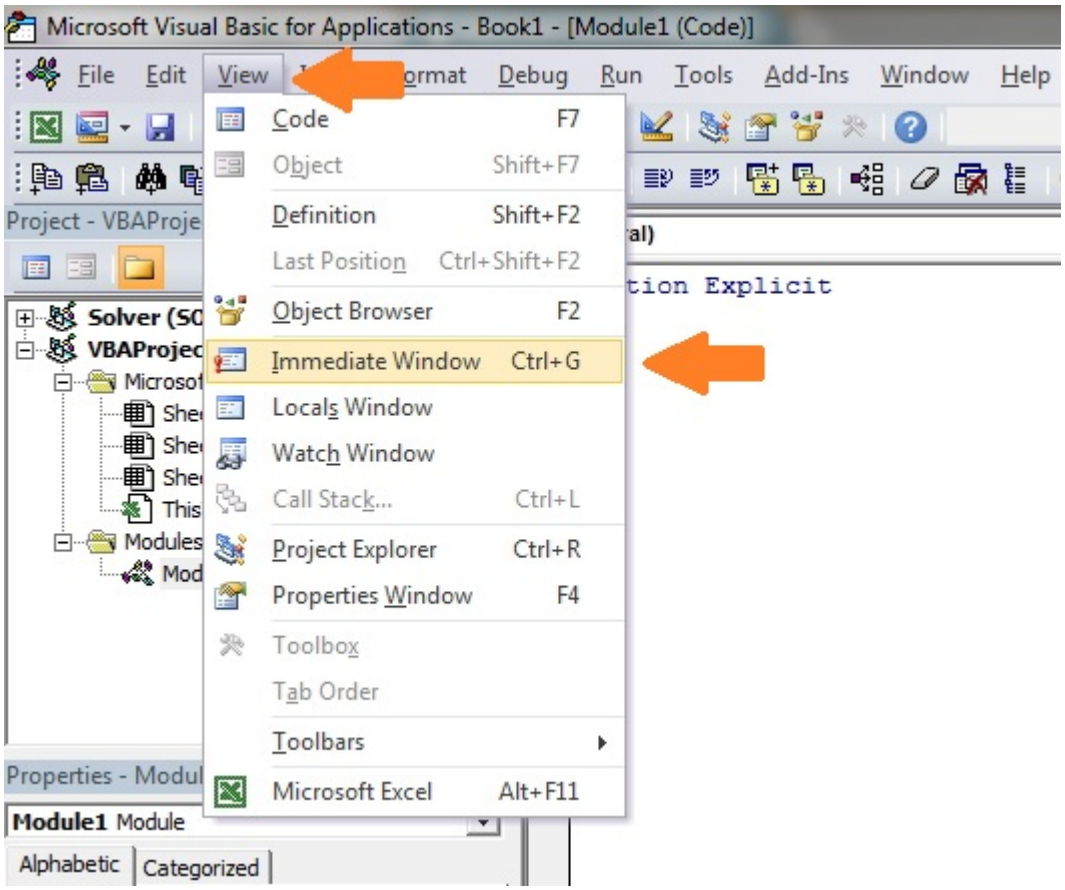
Examples of Using the VBA Workbook

The following examples show what you can do with the workbook.

Note: To try this example create two open workbooks called **Test1.xlsx** and **Test2.xlsx**.

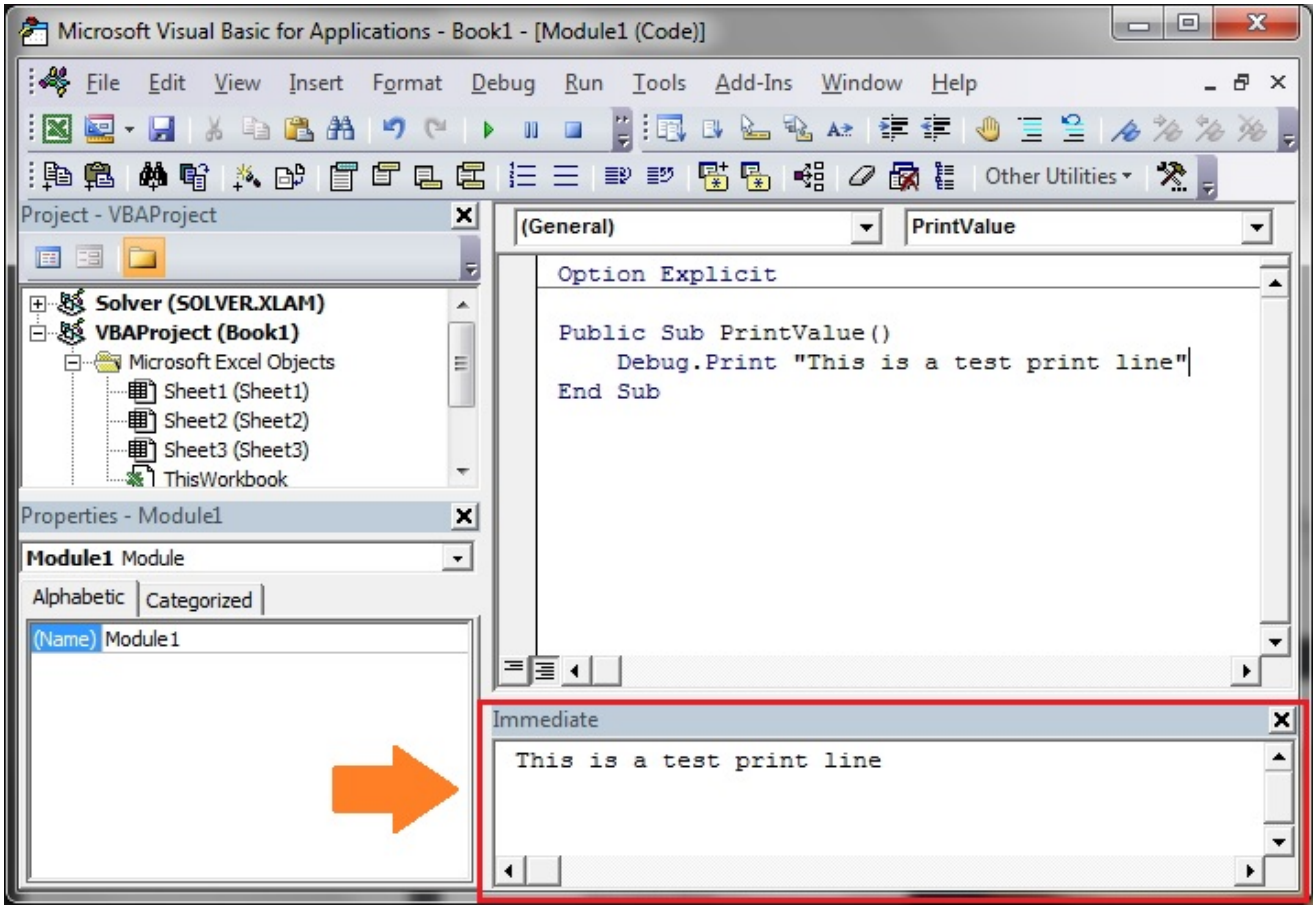
```
Public Sub WorkbookProperties()

    ' Prints the number of open workbooks
    Debug.Print Workbooks.Count

    ' Prints the full workbook name
    Debug.Print Workbooks("Test1.xlsx").FullName

    ' Displays the full workbook name in a message dialog
    MsgBox Workbooks("Test1.xlsx").FullName

    ' Prints the number of worksheets in Test2.xlsx
    Debug.Print Workbooks("Test2.xlsx").Worksheets.Count

    ' Prints the name of currently active sheet of Test2.xlsx
    Debug.Print Workbooks("Test2.xlsx").ActiveSheet.Name

    ' Closes workbook called Test1.xlsx
    Workbooks("Test1.xlsx").Close

    ' Closes workbook Test2.xlsx and saves changes
    Workbooks("Test2.xlsx").Close SaveChanges:=True

End Sub
```

Note: In the code examples I use **Debug.Print** a lot. This function prints values to the Immediate Window. To view this window select View->Immediate Window from the menu(Shortcut is Ctrl + G)



(https://excelmacromastery.com/wp-content/uploads/2014/12/ImmediateWindow2.jpg)

Shares



(https://excelmacromastery.com/wp-content/uploads/2014/12/ImmediateSampeText.jpg)

Accessing the VBA Workbook by Index

You can also use an **Index** number with **Workbooks()**. The index refers to the order the Workbook was open or created.

Workbooks(1) refers to the workbook that was opened first. Workbooks(2) refers to the workbook that was opened second and so on.

```
' First workbook that was opened
Debug.Print Workbooks(1).Name

' Third workbook that was opened
Debug.Print Workbooks(3).Name

' The last workbook that was opened
Debug.Print Workbooks(Workbooks.Count).Name
```

In this example, we used **Workbooks.Count**. This is the number of workbooks that are currently in the Workbooks collection. That is, the number of workbooks currently open. So using it as the Index gives us the last workbook that was opened

Using the index is not really useful unless you really need to know the order. For this reason, you should avoid using it. You should use the workbook name with **Workbooks()** instead.

Finding all Open Workbooks

Sometimes you may want to access all the workbooks that are open. In other words, all the items in the **Workbooks()** collection.

You can do this using the For Each (http://excelmacromastery.com/the-ultimate-guide-to-loops-in-excel-vba/#The_For_Each_Loop) loop.

```
Public Sub PrintWrkFileName()

    ' Prints out the full filename of all open workbooks
    Dim wrk As Workbook
    For Each wrk In Workbooks
        Debug.Print wrk.FullName
    Next wrk

End Sub
```

You can also use the standard For (http://excelmacromastery.com/the-ultimate-guide-to-loops-in-excel-vba/#The_For_Loop)loop to access all the open workbooks

```
Public Sub PrintWrkFileNameIdx()

    ' Prints out the full filename of all open workbooks
    Dim i As Long
    For i = 1 To Workbooks.Count
        Debug.Print Workbooks(i).FullName
    Next i

End Sub
```

For accessing workbooks either of these methods is fine. The For Each (http://excelmacromastery.com/the-ultimate-guide-to-loops-in-excel-vba/#The_For_Each_Loop) loop is generally preferred when you are accessing a large number of objects. In terms of open workbooks this is rarely an issue.

Note: Both examples read in the order of the first opened to the last opened. If you want to read in reverse order(last to first) you can do this

```
Public Sub PrintWrkFileNameIdxRev()

    ' Prints out the full filename of all open workbooks
    ' in reverse order.
    Dim i As Long
    For i = Workbooks.Count To 1 Step -1
        Debug.Print Workbooks(i).FullName
    Next i

End Sub
```

Open Workbook

So far we have dealt with workbooks that are already open. Of course, having to manually open a workbook before running a Macro, defeats the purpose of automating tasks. The **Open Workbook** task should be performed by VBA.

The following VBA code opens the workbook *"Book1.xlsm"* in the *"C:\Docs"* folder

```
Public Sub OpenWrk()

    ' Open the workbook and print the number of sheets it contains
    Workbooks.Open ("C:\Docs\Book1.xlsm")

    Debug.Print Workbooks("Book1.xlsm").Worksheets.Count

    ' Close the workbook without saving
    Workbooks("Book1.xlsm").Close saveChanges:=False

End Sub
```

It is a good idea to check a workbook actually exists before you try to open it. This will prevent you getting errors. The **Dir** function allows you to easily do this .

```
Public Sub OpenWrkDir()

    If Dir("C:\Docs\Book1.xlsm") = "" Then
        ' File does not exist - inform user
        MsgBox "Could not open the workbook. Please check it exists"
    Else
        ' open workbook and do something with it
        Workbooks.Open("C:\Docs\Book1.xlsm").Open
    End If

End Sub
```

Check For Open Workbook

If you are opening a workbook as **Read-Only**, it doesn't matter if it is already open. However, if you're going to update data in a workbook then it is a good idea to check if it is already open.

The function below can be used to check if the workbook is currently open. If not, then it will open the workbook. In either case you will end up with the workbook opened.

(The code below is taken from this StackOverFlow entry (http://stackoverflow.com/questions/9373082/detect-whether-excel-workbook-is-already-open))

```
Function GetWorkbook(ByVal sFullFilename As String) As Workbook

    Dim sFilename As String
    sFilename = Dir(sFullFilename)

    On Error Resume Next
    Dim wk As Workbook
    Set wk = Workbooks(sFilename)

    If wk Is Nothing Then
        Set wk = Workbooks.Open(sFullFilename)
    End If

    On Error Goto 0
    Set GetWorkbook = wk

End Function
```

You can use this function like this

```
Sub ExampleOpenWorkbook()

    Dim sFilename As String
    sFilename = "C:\Docs\Book2.xlsx"

    Dim wk As Workbook
    Set wk = GetWorkbook(sFilename)

End Sub
```

This code is fine is most situations. However, if the workbook could be currently open in read-only mode or could be currently opened by another user then you may want to use a slightly different approach.

An easy way to deal this with this scenario is to insist that the file must be closed for the application to run successfully. You can use the function below to simply check is the file already open and if so inform the user that it must be closed first.

(The code below is also taken from this StackOverFlow entry (http://stackoverflow.com/questions/9373082/detect-whether-excel-workbook-is-already-open))

```
Function IsWorkBookOpen(FileName As String) As Boolean

    Dim ff As Long, ErrNo As Long

    On Error Resume Next

    ' Open file and store error number
    ff = FreeFile()
    Open FileName For Input Lock Read As #ff
    Close ff
    ErrNo = Err
    On Error Goto 0

    ' Check errro number
    Select Case ErrNo

        Case 0 ' No error
            IsWorkBookOpen = False
        Case 70 ' Permission denied error
            IsWorkBookOpen = True
        Case Else ' Other error
            Error ErrNo

    End Select

End Function
```

An example of using this function is shown below. In this case, if the workbook is already open then you inform the user that is must be closed for the macro to proceed.

```
Sub ExampleUse()

    Dim sFilename As String
    sFilename = "C:\temp\Book1.xlsm"

    If IsWorkBookOpen(sFilename) = True Then
        MsgBox "File is already open. Please close file and run macro again."
        Exit Sub
    End If

    ' Write to workbook here

End Sub
```

Close Workbook

To Close a Workbook in Excel VBA is very simple. You simply call the **Close** method of the workbook.

```
wk.Close
```

Normally when you close a workbook in VBA, you don't want to see messages from Excel asking if you want to save the file.

You can specify whether to save the workbook or not and then the Excel messages will not appear.

```
' Don't save changes
wk.Close SaveChanges:= False

' Do save changes
wk.Close SaveChanges:= True
```

Obviously, you cannot save changes to a workbook that is currently open as read-only.

Save Workbook

We have just seen that you can save a workbook when you close it. If you want to save it any other stage you can simply use the **Save** method

```
wk.Save
```

You can also use the **SaveAs** method

```
wk.SaveAs "C:\Backups\accounts.xlsx"
```

The Workbook SaveAs method comes with twelve parameters which allow you to add a password, set the file as read-only and so on. You can see the details here (https://msdn.microsoft.com/en-us/library/office/ff841185.aspx)

You can also use VBA to save the workbook as a copy using **SaveCopyAs**

```
wk.SaveCopyAs "C:\Docs\Copy.xlsm"
```

Copy Workbook

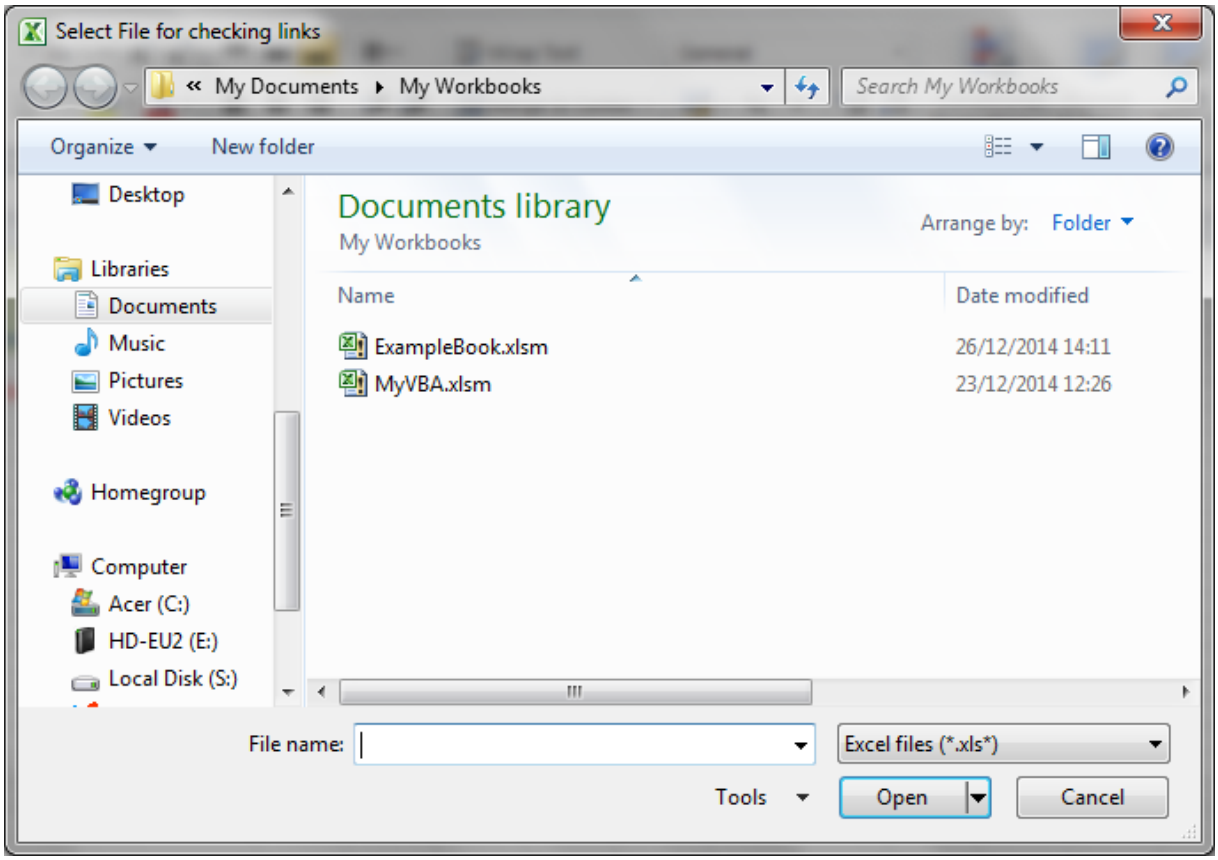
If the workbook is open you can use the two methods in the above section to create a copy i.e. **SaveAs** and **SaveCopyAs**.

If you want to copy a workbook without opening it then you can use **FileCopy** as the following example demonstrates

```
Public Sub CopyWorkbook()
    FileCopy "C:\Docs\Docs.xlsm", "C:\Docs\Example_Copy.xlsm"
End Sub
```

Using the File Dialog To Open a Workbook

The previous section shows you how to open a workbook with a given name. Sometimes you may want the user to select the workbook. You can easily use the **Windows File Dialog** shown here



(https://excelmacromastery.com/wp-content/uploads/2014/12/FileDialog-Workbooks.png)

The Windows File Dialog

The following function opens a workbook using the File Dialog. The function returns the full file name if a file was selected. If the user cancels it displays a message and returns an empty string.

```
Public Function UserSelectWorkbook() As String

    On Error Goto ErrorHandler

    Dim sWorkbookName As String

    Dim FD As FileDialog
    Set FD = Application.FileDialog(msoFileDialogFilePicker)

    ' Open the file dialog
    With FD
        ' Set Dialog Title
        .Title = "Please Select File"

        ' Add filter
        .Filters.Add "Excel Files", "*.xls;*.xlsx;*.xlsm"

        ' Allow selection of one file only
        .AllowMultiSelect = False

        ' Display dialog
        .Show

        If FD.SelectedItems.Count > 0 Then
            UserSelectWorkbook = FD.SelectedItems(1)
        Else
            MsgBox "Selecting a file has been cancelled. "
            UserSelectWorkbook = ""
        End If
    End With

    ' Clean up
    Set FD = Nothing
Done:
    Exit Function
ErrorHandler:
    MsgBox "Error: " + Err.Description
End Function
```

When you call this function you have to check for the user cancelling the dialog. The following example shows you how to easily call the **UserSelectWorkbook** function and handle the case of the user cancelling

```
Public Sub TestUserSelect()

    Dim userBook As Workbook, sFilename As String

    ' Call the UserSelectworkbook function
    sFilename = UserSelectWorkbook()

    ' If the filename returns is blank the user cancelled
    If sFilename <> "" Then
        ' Open workbook and do something with it
        Set userBook = Workbooks.Open(sFilename)
    End If

End Sub
```

You can customise the dialog by changing the Title, Filters and AllowMultiSelect in the **UserSelectWorkbook** function.

Using ThisWorkbook

There is an easier way to access the current workbook than using **Workbooks()**. You can use the keyword ***ThisWorkbook***. It refers to the current workbook i.e. the workbook that contains the VBA code.

If our code is in a workbook call MyVBA.xlsm then **ThisWorkbook** and **Workbooks("MyVBA.xlsm")** refer to the same workbook.

Using **ThisWorkbook** is more useful than using **Workbooks()**. With ThisWorkbook we do not need to worry about the name of the file. This gives us two advantages:

1. Changing the filename will not affect the code
2. Copying the code to another workbook will not require a code change

These may seem like very small advantages. The reality is your filenames will change all the time. Using **ThisWorkbook** means your code will still work fine.

The following example shows two lines of code. One using **ThisWorkbook** and one using **Workbooks()**. The one using Workbooks will no longer work if the name of MyVBA.xlsm changes.

```
Public Sub WriteToCellUsingThis()

    ' Both lines do the same thing.
    Debug.Print ThisWorkbook.FullName
    Debug.Print Workbooks("MyVBA.xlsm").FullName

End Sub
```

Using the ActiveWorkbook

ActiveWorkbook refers to the workbook that is currently active. This is the one that the user last clicked on.

This can seem useful at first. The problem is that any workbook can become active by a simple mouse click. This means you could easily write data to the wrong workbook.

Using **ActiveWorkbook** also makes the code hard to read. It may not be obvious from the code which workbook should be the active one.

I hope I made it clear that you should avoid using **ActiveWorkbook** unless you really have to. In this case be very careful.

Examples of the Accessing the Workbook

We've looked at all the ways of accessing a workbook. The following code shows examples of these ways

```
Public Sub WorkbooksUse()

    ' This is a workbook that is already open and called MyVBA.xlsm
    Debug.Print Workbooks("MyVBA.xlsm").FullName

    ' The workbook that contains this code
    Debug.Print ThisWorkbook.FullName

    ' The open workbook that was opened first
    Debug.Print Workbooks(1).FullName

    ' The open workbook that was opened last
    Debug.Print Workbooks(Workbooks.Count).FullName

    ' The workbook that is the currently active one
    Debug.Print ActiveWorkbook.FullName

    ' No workbook mentioned - the active one will be used
    Debug.Print Worksheets("Sheet1").Name

    ' A closed workbook called Book1.xlsm in folder C:\Docs
    Workbooks.Open ("C:\Docs\Book1.xlsm")
    Debug.Print Workbooks("Book1.xlsm").FullName
    Workbooks("Book1.xlsm").Close

End Sub
```

Declaring a VBA Workbook variable

The reason for declaring a workbook variable is to make your code easier to read and understand. It is easier to see the advantage using an example

```
Public Sub OpenWrkObjects()

    Dim wrk As Workbook
    Set wrk = Workbooks.Open("C:\Docs\Book1.xlsm")

    ' Print number of sheets in each book
    Debug.Print wrk.Worksheets.Count
    Debug.Print wrk.Name

    wrk.Close

End Sub
```

You can set a workbook variable with any of the access methods we have seen.

The following shows you the same code without a workbook variable

```
Public Sub OpenWrkNoObjects()

   Workbooks.Open ("C:\Docs\Book1.xlsm")

   Debug.Print Workbooks("Book2.xlsm").Worksheets.Count
   Debug.Print Workbooks("Book2.xlsm").Name

    Workbooks("Book2.xlsm").Close

End Sub
```

In these examples the difference is not major. However, when you have a lot of code, using a variable is useful particularly for worksheet and ranges where the names tend to be long e.g. **thisWorkbook.Worksheets("Sheet1").Range("A1")**.

You can name the workbook variable to be something like wrkRead or wrkWrite. Then at a glance you can see what this workbook is being used for.

Create New Workbook

To create a new workbook you use the Workbooks **Add** function. This function creates a new blank workbook. It is the same as selecting New Workbook from the Excel File menu.

When you create a new workbook you will generally want to Save it. The following code shows you how to do this.

```
Public Sub AddWordbook()

    Dim wrk As Workbook
    Set wrk = Workbooks.Add

    ' Save as xlsx. This is the default.
    wrk.SaveAs "C:\Temp\Example.xlsx"

    ' Save as a Macro enabled workbook
    wrk.SaveAs "C:\Temp\Example.xlsm", xlOpenXMLWorkbookMacroEnabled

End Sub
```

When you create a new workbook it normally contains three sheets. This is determined by the property **Application.SheetsInNewWorkbook**.

If you want to have a different number of sheets in a new workbook then you change this property before you create the new workbook. The following example shows you how to create a new workbook with seven sheets.

```
Public Sub AddWordbookMultiSheets()

    ' Store SheetsInNewWorkbook value so we can reset it later
    Dim sheetCnt As Long
    sheetCnt = Application.SheetsInNewWorkbook

    ' Set sheets in a new workbook to be 7
    Application.SheetsInNewWorkbook = 7

    ' Workbook will be created with 7 sheets
    Dim wrk As Workbook
    Set wrk = Workbooks.Add

    ' Display sheet count
    Debug.Print "number of sheets: " & CStr(wrk.Worksheets.Count)

    ' Reset to original value
    Application.SheetsInNewWorkbook = sheetCnt

End Sub
```

The With keyword and the Workbook

The **With** keyword makes reading and writing VBA code easier. Using **With** means you only need to mention the item once. **With** is used with **Objects.** These are items such as Workbooks, Worksheets (http://excelmacromastery.com/the-complete-guide-to-worksheets-in-excel-vba/)and Ranges (http://excelmacromastery.com/the-complete-guide-to-ranges-and-cells-in-excel-vba/).

The following example has two Subs. The first is similar to code we have seen so far. The second uses the **With** keyword. You can see the code is much clearer in the second Sub. The keywords **End With** mark the finish of a section code using **With**.

```
' Not using the With keyword
Public Sub NoUsingWith()

   Debug.Print Workbooks("Book2.xlsm").Worksheets.Count
   Debug.Print Workbooks("Book2.xlsm").Name
   Debug.Print Workbooks("Book2.xlsm").Worksheets(1).Range("A1")
   Workbooks("Book2.xlsm").Close

End Sub

' Using With makes the code easier to read
Public Sub UsingWith()

    With Workbooks("Book2.xlsm")
        Debug.Print .Worksheets.Count
        Debug.Print .Name
        Debug.Print .Worksheets(1).Range("A1")
        .Close
    End With

End Sub
```

Summary

The following is a brief summary of the main points of this post

1. To get the workbook containing the code use **ThisWorkbook**.
2. To get any open workbook use **Workbooks(“Example.xlsx”)**.
3. To open a workbook use **Set Wrk = Workbooks.Open(“C:\Folder\Example.xlsx”).**
4. Allow the user to select a file using the **UserSelectWorkbook** function provided above.

5. To create a copy of an open workbook use the **SaveAs** property with a filename.
6. To create a copy of a workbook without opening use the **FileCopy** function.
7. To make your code easier to read and write use the **With** keyword.
8. Another way to make your code clear is to use a **Workbook** variables
9. To run through all open Workbooks use **For Each wk in Workbooks** where **wk** is a workbook variable.
10. Try to avoid using **ActiveWorkbook** and **Workbooks(Index)** as their reference to a workbook is temporary.

You can see a quick guide to the topic at the top of this post

Conclusion

This was an in-depth post about a very important element of VBA – the Workbook. I hope you found it beneficial. Excel is great at providing many ways to perform similar actions but the downside is it can lead to confusion at times.

To get the most benefit from this post I recommend you try out the examples. Create some workbooks and play around with the code. Make changes to the code and see how the changes affect the outcome. Practice is the best way to learn VBA.

If you found this post useful then feel free to share it with others. You may also want to check out The Complete Guide to Worksheets in Excel VBA (http://excelmacromastery.com/the-complete-guide-to-worksheets-in-excel-vba/). You can view all the posts by category here (http://excelmacromastery.com/a-quick-guide-to-the-vba-posts/).

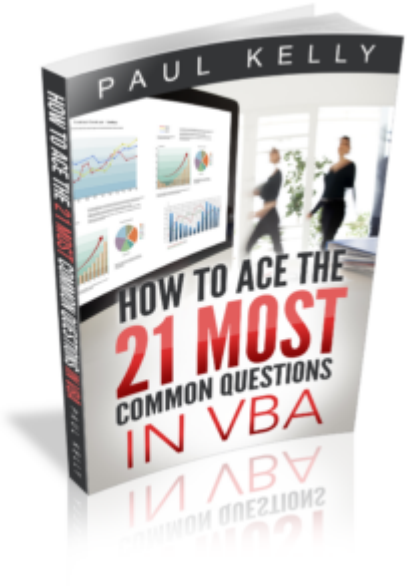
What's Next

Once you understand Workbooks the next topics you may want to check out are Worksheets (http://excelmacromastery.com/the-complete-guide-to-worksheets-in-excel-vba/) and Ranges and Cells (http://excelmacromastery.com/the-complete-guide-to-ranges-and-cells-in-excel-

vba/). These three topics are a core part of VBA and it's vital to understand them. You can get the **complete list of all the VBA posts** here (http://excelmacromastery.com/a-quick-guide-to-the-vba-posts/).

I also have a free eBook(see below) which you will find useful if you are new to VBA.

Get the Free eBook



(https://excelmacromastery.leadpages.co/leadbox/14791da73f72a2%3A106f25298346dc/5636318

Please feel free to subscribe to my newsletter and get exclusive VBA content that you cannot find here on the blog, as well as free access to my eBook, **How to Ace the 21 Most Common Questions in VBA** which is full of examples you can use in your own code.

DOWNLOAD NOW

(https://excelmacromastery.leadpages.co/leadbox/14791da73f72a2%3A106f25298346dc/5636318

(https://excelmacromastery.leadpages.co/leadbox/14791da73f72a2%3A106f25298346dc/5636318

Create Workbook (https://excelmacromastery.com/tag/create-workbook/) ForEach (https://excelmacromastery.com/tag/foreach/) Open Workbook

(https://excelmacromastery.com/tag/open-workbook/) Range (https://excelmacromastery.com/tag/range/) Workbooks (https://excelmacromastery.com/tag/workbooks/) Workbooks Open (https://excelmacromastery.com/tag/workbooks-open/) Worksheets (https://excelmacromastery.com/tag/worksheets/)

Next
The Complete Guide To The VBA Worksheet (https://excelmacromastery.com/excel-vba-worksheet/)

10 COMMENTS

Sidharth Saini

February 14, 2016 at 5:13 pm (https://excelmacromastery.com/excel-vba-workbook/#comment-2)

Very nicely explained topics. But symbols of > and < in examples are have problems. Please change them.

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=2#respond)

Paul Kelly

February 15, 2016 at 3:26 pm (https://excelmacromastery.com/excel-vba-workbook/#comment-3)

Hi Sidharth,

Thanks for pointing that out.
Sometimes when I update a post, WordPress automatically changes the greater/less than symbols. I've updated the post to fix them.

Regards
Paul

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=3#respond)

Pooja

February 18, 2016 at 12:01 pm (https://excelmacromastery.com/excel-vba-workbook/#comment-4)

Hi Paul, I've gone through couple of websites to actually learn Macros if a person is novice and none could provide the basics so clearly the way you are doing.
I'm actually learning VBA and macros through your website.
Gr8 work 🙂

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=4#respond)

Paul Kelly

February 18, 2016 at 1:45 pm (https://excelmacromastery.com/excel-vba-workbook/#comment-5)

Thanks Pooja, Glad you like it.

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=5#respond)

Petros

June 2, 2016 at 8:27 pm (https://excelmacromastery.com/excel-vba-workbook/#comment-6)

Paul, what is your email please?

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=6#respond)

Paul Kelly

June 3, 2016 at 7:23 am (https://excelmacromastery.com/excel-vba-workbook/#comment-7)

Hi Petros

You can email me at: paul@ExcelMacroMastery.com (mailto:paul@ExcelMacroMastery.com)

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=7#respond)

Cor

August 24, 2016 at 6:12 am (https://excelmacromastery.com/excel-vba-workbook/#comment-8)

Hi Paul,

I like your site because the explanations you provide are really making me understand some things I didn't before.

However I found a few hickups in the 'Finding all open workbooks section'. In the seconde and third example (PrintWrkFileNameIdx and PrintWrkFileNameIdxRev) you use "Dim i" and then continue with "'Next wrk". I guess someone copied the first example (PrintWrkFileName), where "wrk" was actually used, and changed the contents 😉

Greetings

Cor

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=8#respond)

Paul Kelly

August 24, 2016 at 9:01 am (https://excelmacromastery.com/excel-vba-workbook/#comment-9)

Hi Cor,

Thanks for your comment. Glad you like the site. I've update the post to fix those issues.

Regards
Paul

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=9#respond)

sneha sheth

January 4, 2017 at 12:23 pm (https://excelmacromastery.com/excel-vba-workbook/#comment-2275)

hi paul….
what a great piece of work. i m glad i accessed ur site. being a software engineer my self … i was really in search of logically connected concepts and not the syntax ….and i found both… thanks a lot

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=2275#respond)

Paul Kelly

January 4, 2017 at 2:00 pm (https://excelmacromastery.com/excel-vba-workbook/#comment-2278)

Thanks Sneha, glad you like it.

Reply (https://excelmacromastery.com/excel-vba-workbook/?replytocom=2278#respond)

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Post Comment