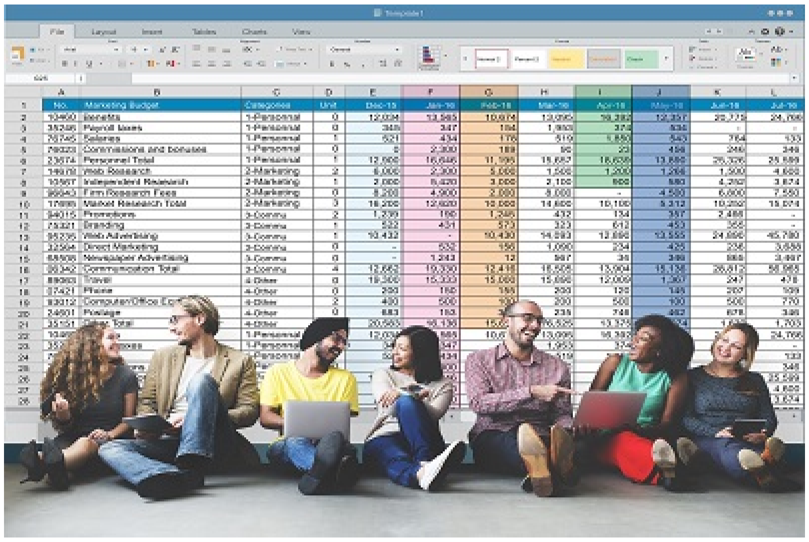
EXCEL MACRO MASTERY (HTTPS://EXCELMACROMASTERY.COM/)

THE MISSING VBA HANDBOOK

The Complete Guide To The VBA Worksheet

JANUARY 2, 2015 (HTTPS://EXCELMACROMASTERY.COM/EXCEL-VBA-WORKSHEET/) BY PAUL KELLY (HTTPS://EXCELMACROMASTERY.COM/AUTHOR/ADMIN/) · 14 COMMENTS (HTTPS://EXCELMACROMASTERY.COM/EXCEL-VBA-WORKSHEET/#COMMENTS)

Shares



"The visionary starts with a clean sheet of paper, and re-imagines the world" – Malcolm Gladwell

This post provides a complete guide to using the Excel **VBA Worksheet** in Excel VBA. If you want to know how to do something quickly then check out the quick guide to the VBA Worksheet below.

If you are new to VBA then this post is a great place to start. I like to break things down into simple terms and explain them in plain English without the jargon.

You can **read through the post from start to finish** as it is written in a logical order. If you prefer, you can use the table of contents below and go directly to the topic of your choice.

Contents [hide]

adbox/145db6b73f72a2%3A106f25298346dc/5676073085829120/)

A Quick Guide to the VBA Worksheet

The following table gives a quick run down to the different worksheet methods.

oks-

Note: I use *Worksheets* in the table below without specifying the workbook i.e. *Worksheets* rather than *ThisWorkbook.Worksheets*, *wk.Worksheets* etc. This is to make the examples clear and easy to read. You should always specify the workbook when using *Worksheets*. Otherwise the active workbook will be used by default.

Task	**How to**
Access worksheet by name	Worksheets("Sheet1")

Task	**How to**

Access worksheet by position from left	Worksheets(2)
	Worksheets(4)

Access the left most worksheet	Worksheets(1)
Access the right most worksheet	Worksheets(Worksheets.Count)
Access using worksheet code name(current workbook only)	see Code Name section below
Access using worksheet code name(other workbook)	see Code Name section below
Access the active worksheet	ActiveSheet
Declare worksheet variable	**Dim** sh **As** Worksheet
Assign worksheet variable	**Set** sh = Worksheets("Sheet1")
Add worksheet	Worksheets.Add
Add worksheet and assign to variable	**Set** sh =Worksheets.Add
Add worksheet to first position(left)	Worksheets.Add Before:=Worksheets(1)
Add worksheet to last position(right)	Worksheets.Add after:=Worksheets(Worksheets.Count)
Add multiple worksheets	Worksheets.Add Count:=3
Activate Worksheet	sh.Activate
Copy Worksheet	sh.Copy
Copy after a worksheet	sh1.Copy After:=Sh2
Copy before a worksheet	sh1.Copy Before:=Sh2
Delete Worksheet	sh.Delete

omment-
omment-

Task	How to

Delete Worksheet without warning	Application.DisplayAlerts = **False**
	sh.Delete
	Application.DisplayAlerts = **True**

Change worksheet name	sh.Name = "Data"

Show/hide worksheet	sh.Visible = xlSheetHidden
	sh.Visible = xlSheetVisible

Loop through all worksheets(For)	**Dim** i **As Long** **For** i = 1 **To** Worksheets.Count **Debug.Print** Worksheets(i).Name **Next** i
Loop through all worksheets(For Each)	**Dim** sh **As** Worksheet **For Each** sh **In** Worksheets **Debug.Print** sh.Name **Next**

Introduction

The three most important elements of VBA are the Workbook (http://excelmacromastery.com/excel-vba-workbook/), the Worksheet and Cells (http://excelmacromastery.com/excel-vba-range-cells). Of all the code your write, 90% will involve one or all of them.

The most **common use of the worksheet** in VBA is for accessing its cells. You may use it to protect, hide, add, move or copy a worksheet. However, you will mainly use it to perform some action on one or more cells on the worksheet.

Using Worksheets is more straightforward than using workbooks. With workbooks you may need to open them, find which folder they are in, check if they are in use and so on. With a worksheet, it either exists in the workbook or it doesn't.

Accessing the Worksheet

In VBA, each workbook has a collection of worksheets. There is an entry in this collection for each worksheet in the workbook. This collection is simply called **Worksheets** and is used in a very similar way to the **Workbooks** collection. To get access to a worksheet all you have to do is supply the name.

The code below writes "Hello World" in Cell A1 of Sheet1, Sheet2 and Sheet3 of the current workbook.

```
Public Sub WriteToCell1()

    ' Write To cell A1 In Sheet1,Sheet2 And Sheet3
    ThisWorkbook.Worksheets("Sheet1").Range("A1") = "Hello World"
    ThisWorkbook.Worksheets("Sheet2").Range("A1") = "Hello World"
    ThisWorkbook.Worksheets("Sheet3").Range("A1") = "Hello World"

End Sub
```

The **Worksheets** collection is always belong to a workbook. If we don't specify the workbook then the active workbook is used by default.

```
Public Sub WriteToCell1()

    ' Worksheets refers to the worksheets in the active workbook
    Worksheets("Sheet1").Range("A1") = "Hello World"
    Worksheets("Sheet2").Range("A1") = "Hello World"
    Worksheets("Sheet3").Range("A1") = "Hello World"

End Sub
```

Hide Worksheet

The following examples show how to hide and unhide a worksheet

```
ThisWorkbook.Worksheets("Sheet1").Visible = xlSheetHidden

ThisWorkbook.Worksheets("Sheet1").Visible = xlSheetVisible
```

If you want to prevent a user accessing the worksheet, you can make it “very hidden”. This means it can only be made visible by the code.

```
' Hide from user access
ThisWorkbook.Worksheets("Sheet1").Visible = xlVeryHidden

' This is the only way to make a xlVeryHidden sheet visible
ThisWorkbook.Worksheets("Sheet1").Visible = xlSheetVisible
```

Protect Worksheet

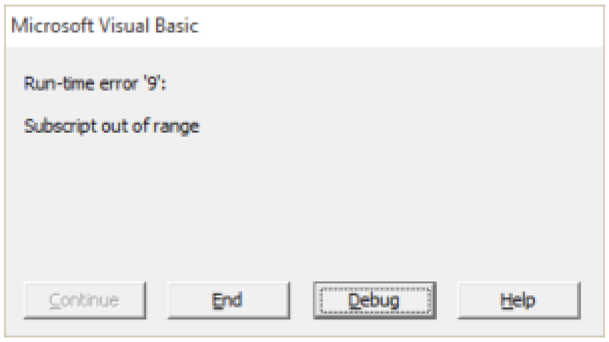
Another example of using the worksheet is when you want to protect it

```
ThisWorkbook.Worksheets("Sheet1").Protect Password:="MyPass"

ThisWorkbook.Worksheets("Sheet1").Unprotect Password:="MyPass"
```

Subscript Out of Range

When you use *Worksheets* you may get the error:

Run-time Error 9 Subscript out of Range


Microsoft Visual Basic

Run-time error '9':

Subscript out of range

This means you tried to access a worksheet that doesn't exist. This may happen for the following reasons

1. The worksheet name given to *Worksheets* is spelled incorrectly.
2. The name of the worksheet has changed.
3. The worksheet was deleted.
4. The index was to large e.g. You used *Worksheets(5)* but there are only four worksheets
5. The wrong workbook is being used e.g. *Workbooks(**"book1.xlsx"**).Worksheets("Sheet1")* instead of *Workbooks(**"book3.xlsx"**).Worksheets("Sheet1")*.

If you still have issues then use one of the loops from Loop Through The Worksheets section to print the names of all worksheets the collection.

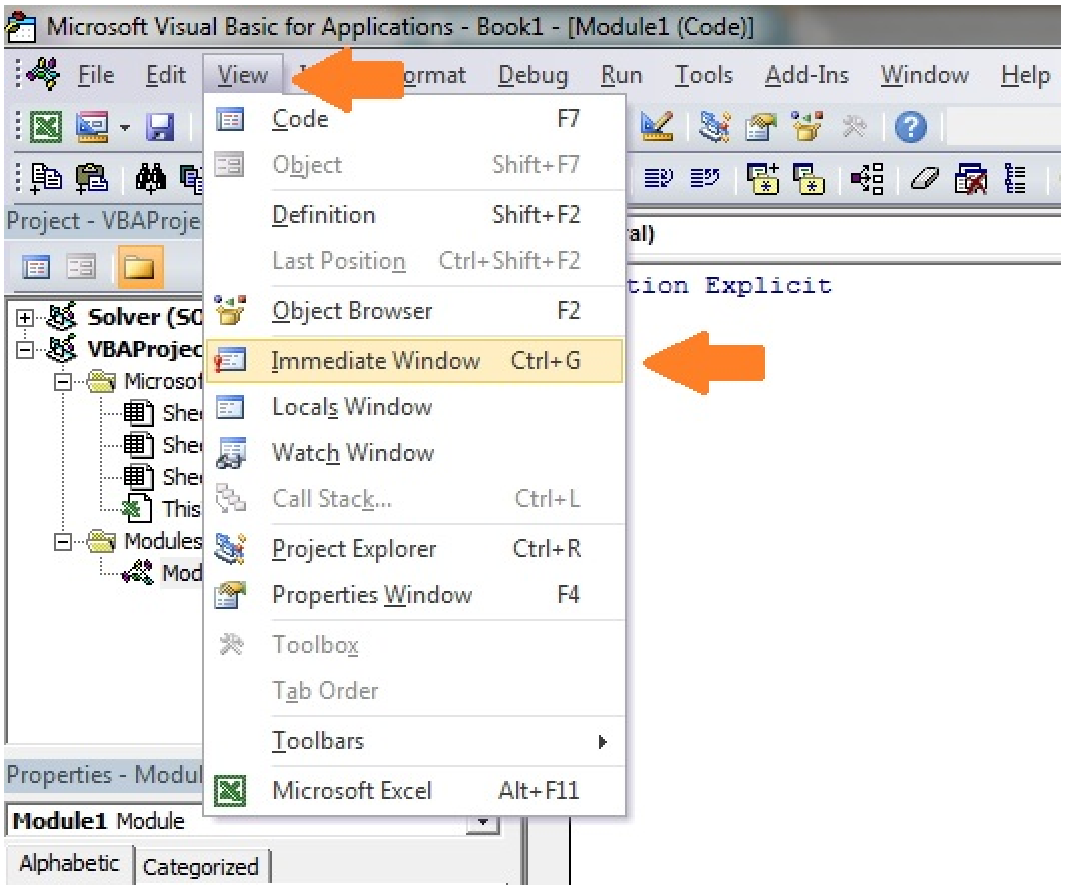
Using the Index to Access the Worksheet

So far we have been using the sheet name to access the sheet. The index refers to the sheet tab position in the workbook. As the position can easily be changed by the user it is not a good idea to use this.

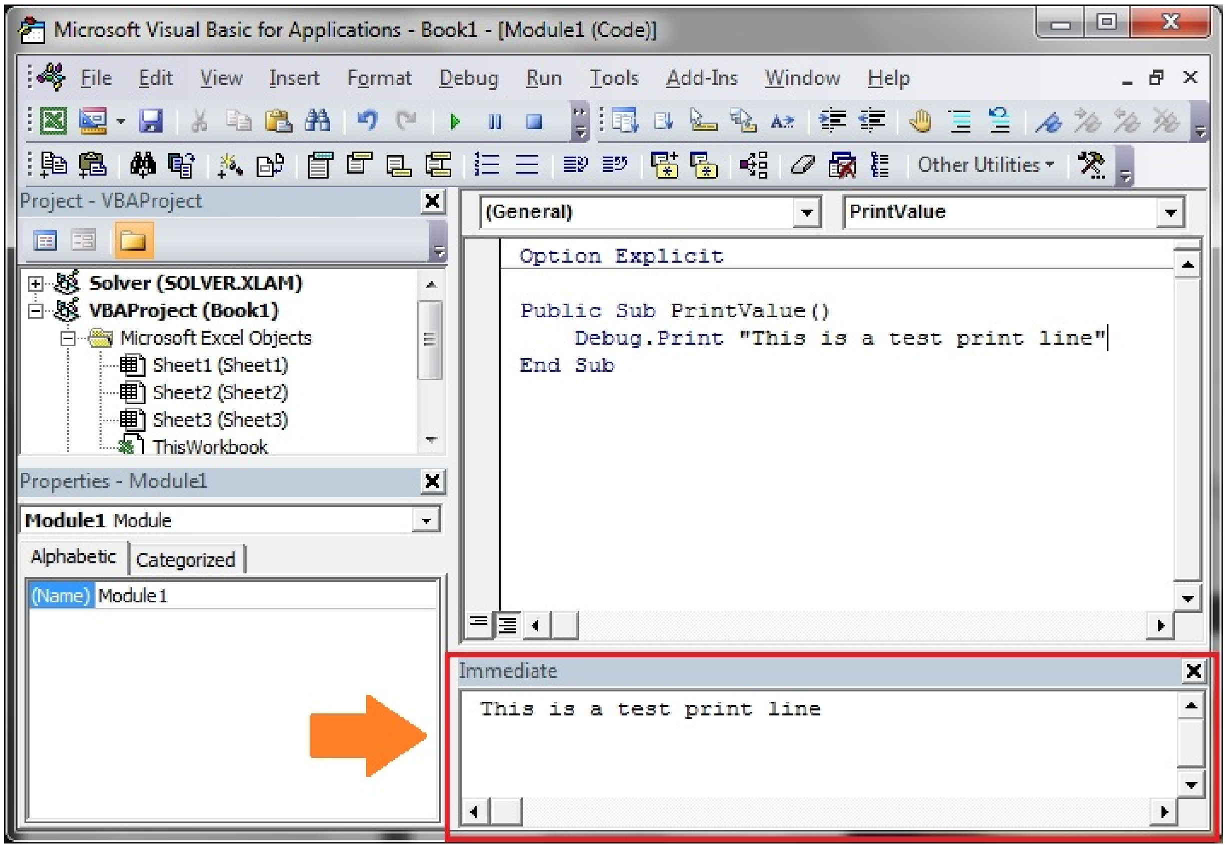
The following code shows examples of using the index

```
' Using this code is a bad idea as
' sheet positions changes all the time
Public Sub UseSheetIdx()

    With ThisWorkbook
        ' Left most sheet
        Debug.Print .Worksheets(1).Name
        ' The third sheet from the left
        Debug.Print .Worksheets(3).Name
        ' Right most sheet
        Debug.Print .Worksheets(.Worksheets.Count).Name
    End With

End Sub
```

In the example above, I used **Debug.Print** to print to the Immediate Window. To view this window select View->Immediate Window(or Ctrl G)



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



```
Option Explicit

Public Sub PrintValue()
    Debug.Print "This is a test print line"
End Sub
```

```
This is a test print line
```

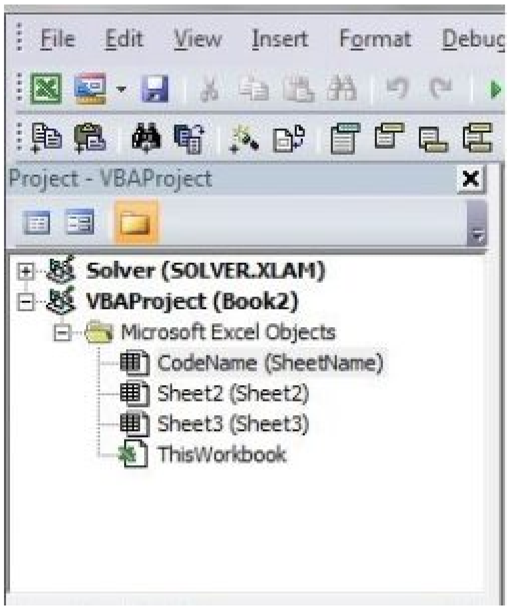
(https://excelmacromastery.com/wp-content/uploads/2014/12/ImmediateSampeText.jpg)

Using the Code Name of a Worksheet

The best method of accessing the worksheet is using the code name. Each worksheet has a sheet name and a code name. The sheet name is the name that appears in the worksheet tab in Excel.

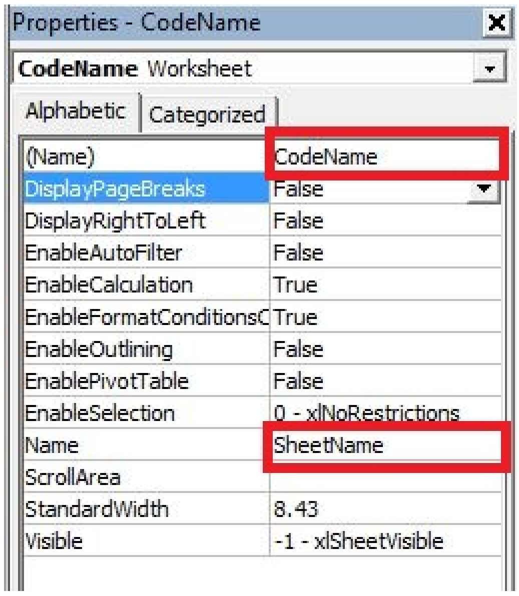
Changing the sheet name does not change the code name meaning that referencing a sheet by the code name is a good idea.

If you look in the VBE property window you will see both names. In the image you can see that the code name is the name outside the parenthesis and the sheet name is in the parenthesis.



(https://excelmacromastery.com/wp-content/uploads/2014/12/VBEProperties1.jpg)

You can change both the sheet name and the code name in the property window of the sheet(see image below).


Properties - CodeName

CodeName Worksheet

Alphabetic | Categorized

(Name)	CodeName
DisplayPageBreaks	False
DisplayRightToLeft	False
EnableAutoFilter	False
EnableCalculation	True
EnableFormatConditionsC	True
EnableOutlining	False
EnablePivotTable	False
EnableSelection	0 - xlNoRestrictions
Name	SheetName
ScrollArea	
StandardWidth	8.43
Visible	-1 - xlSheetVisible

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

If your code refers to the code name then the user can change the name of the sheet and it will not affect your code. In the example below we reference the worksheet directly using the code name.

```
Public Sub UseCodeName2()

    ' Using the code name of the worksheet
    Debug.Print CodeName.Name
    CodeName.Range("A1") = 45
    CodeName.Visible = True

End Sub
```

This makes the code easy to read and safe from the user changing the sheet name.

Code Name in other Workbooks

There is one drawback to using the code name. It can only refer to worksheets in the workbook that contains the code i.e. ThisWorkbook.

However, we can use a simple function to find the code name of a worksheet in a different workbook.

```
Public Sub UseSheet()

    Dim sh As Worksheet
    ' Get the worksheet using the codename
    Set sh = SheetFromCodeName("CodeName", ThisWorkbook)
    ' Use the worksheet
    Debug.Print sh.Name

End Sub

' This function gets the worksheet object from the Code Name
Public Function SheetFromCodeName(Name As String, bk As Workbook) As Worksheet

    Dim sh As Worksheet
    For Each sh In bk.Worksheets
        If sh.CodeName = Name Then
           Set SheetFromCodeName = sh
           Exit For
        End If
    Next sh

End Function
```

Using the above code means that if the user changes the name of the worksheet then your code will not be affected.

There is another way of getting the sheet name of an external workbook using the code name. You can use the **VBProject** element of that Workbook.

You can see how to do this in the example below. I have included this for completeness only and I would recommend using the method in the previous example rather than this one.

```
Public Function SheetFromCodeName2(codeName As String _
                             , bk As Workbook) As Worksheet

    ' Get the sheet name from the CodeName using the VBProject
    Dim sheetName As String
    sheetName = bk.VBProject.VBComponents(codeName).Properties("Name")

    ' Use the sheet name to get the worksheet object
    Set SheetFromCodeName2 = bk.Worksheets(sheetName)

End Function
```

Code Name Summary

The following is a quick summary of using the Code Name

1. The code name of the worksheet can be used directly in the code e.g. *Sheet1.Range*
2. The code name will still work if the worksheet name is changed.
3. The code name can only be used for worksheets in the same workbook as the code.
4. Anywhere you see *ThisWorkbook.Worksheets("sheetname")* you can replace it with the code name of the worksheet.
5. You can use the *SheetFromCodeName* function from above to get the code name of worksheets in other workbooks.

The Active Sheet

The **ActiveSheet** object refers to the worksheet that is currently active. You should only use ActiveSheet if you have a specific need to refer to the worksheet that is active.

Otherwise you should specify the worksheet you are using.

If you use a worksheet method like **Range** and don't mention the worksheet, it will use the active worksheet by default.

```
' Write to Cell A1 in the active sheet
ActiveSheet.Range("A1") = 99

' Active sheet is the default if no sheet used
Range("A1") = 99
```

Declaring a Worksheet Object

Declaring a worksheet object is useful for making your code neater and easier to read.

The next example shows code for updating ranges of cells. The first Sub does not declare a worksheet object. The second sub declares a worksheet object and the code is therefore much clearer.

```
Public Sub SetRangeVals()

    Debug.Print ThisWorkbook.Worksheets("Sheet1").Name
    ThisWorkbook.Worksheets("Sheet1").Range("A1") = 6
    ThisWorkbook.Worksheets("Sheet1").Range("B2:B9").Font.Italic = True
    ThisWorkbook.Worksheets("Sheet1").Range("B2:B9").Interior.Color = rgbRed

End Sub
```

```
Public Sub SetRangeValsObj()

    Dim sht As Worksheet
    Set sht = ThisWorkbook.Worksheets("Sheet1")

    sht.Range("A1") = 6
    sht.Range("B2:B9").Font.Italic = True
    sht.Range("B2:B9").Interior.Color = rgbRed

End Sub
```

You could also use the With keyword with the worksheet object as the next example shows.

```
Public Sub SetRangeValsObjWith()

    Dim sht As Worksheet
    Set sht = ThisWorkbook.Worksheets("Sheet1")

    With sht
        .Range("A1") = 6
        .Range("B2:B9").Font.Italic = True
        .Range("B2:B9").Interior.Color = rgbRed
    End With

End Sub
```

Accessing the Worksheet in a Nutshell

With all the different ways to access a worksheet, you may be feeling overwhelmed or confused. So in this section, I am going to break it down into simple terms

1. If you want to use whichever worksheet is currently active then use ActiveSheet.

```
ActiveSheet.Range("A1") = 55
```

2. If the worksheet is in the same workbook as the code then use the Code Name.

```
Sheet1.Range("A1") = 55
```

3. If the worksheet is in a different workbook then first get workbook and then get the worksheet.

```
' Get workbook
Dim wk As Workbook
Set wk = Workbooks.Open("C:\Docs\Accounts.xlsx", ReadOnly:=True)

' Then get worksheet
Dim sh As Worksheet
Set sh = wk.Worksheets("Sheet1")
```

If you want to protect against the user changing the sheet name then use the **SheetFromCodeName** function from the Code Name section.

```
' Get workbook
Dim wk As Workbook
Set wk = Workbooks.Open("C:\Docs\Accounts.xlsx", ReadOnly:=True)

' Then get worksheet
Dim sh As Worksheet
Set sh = SheetFromCodeName("sheetcodename",wk)
```

Add Worksheet

The examples in this section show you how to add a new worksheet to a workbook. If you do not supply any arguments to the **Add** function then the new worksheet will be placed before the active worksheet.

When you add a Worksheet, it is created with a default name like “Sheet4”. If you want to change the name then you can easily do this using the **Name** property.

The following example adds a new worksheet and changes the name to “Accounts”. If a worksheet with the name “Accounts” already exists then you will get an error.

```
Public Sub AddSheet()

    Dim sht As Worksheet

    ' Adds new sheet before active sheet
    Set sht = ThisWorkbook.Worksheets.Add
    ' Set the name of sheet
    sht.Name = "Accounts"

    ' Adds 3 new sheets before active sheet
    ThisWorkbook.Worksheets.Add Count:=3

End Sub
```

In the previous example, you are adding worksheets in relation to the active worksheet. You can also specify the exact position to place the worksheet.

To do this you need to specify which worksheet the new one should be inserted before or after. The following code shows you how to do this.

```
Public Sub AddSheetFirstLast()

    Dim shtNew As Worksheet
    Dim shtFirst As Worksheet, shtLast As Worksheet

    With ThisWorkbook

        Set shtFirst = .Worksheets(1)
        Set shtLast = .Worksheets(.Worksheets.Count)

        ' Adds new sheet to first position in the workbook
        Set shtNew = Worksheets.Add(Before:=shtFirst)
        shtNew.Name = "FirstSheet"

        ' Adds new sheet to last position in the workbook
        Set shtNew = Worksheets.Add(After:=shtLast)
        shtNew.Name = "LastSheet"

    End With

End Sub
```

Delete Worksheet

To delete a worksheet you simply call the **Delete** member.

```
Dim sh As Worksheet
Set sh = ThisWorkbook.Worksheets("Sheet12")
sh.Delete
```

Excel will display a warning message when you delete a worksheet. If you want to hide this message you can use the code below

```
Application.DisplayAlerts = False
sh.Delete
Application.DisplayAlerts = False
```

There are two issues to watch out for when it comes to deleting worksheets.

If you try to access the worksheet after deleting it you will get the "Subscript out of Range" error we saw in the Accessing the Worksheet section.

```
Dim sh As Worksheet
Set sh = ThisWorkbook.Worksheets("Sheet2")
sh.Delete

' This line will give 'Subscript out of Range' as "Sheet2" does not exist
Set sh = ThisWorkbook.Worksheets("Sheet2")
```

The second issue is when you assign a worksheet variable. If you try to use this variable after the worksheet is deleted then you will get an Automation error like this

Run-Time error -21147221080 (800401a8') Automation Error

If you are using the Code Name of the worksheet rather than a variable, then this will cause Excel to crash rather than give the automation error.

The following example shows how an automation errors occurs

```
sh.Delete

' This line will give Automation error
Debug.Assert sh.Name
```

If you assign the Worksheet variable to a valid worksheet it will work fine

```
sh.Delete

' Assign sh to another worksheet
Set sh = Worksheets("sheet3")

' This line will work fine
Debug.Assert sh.Name
```

Loop Through the Worksheets

The Worksheets member of Workbooks is a collection of worksheets belonging to a workbook. You can go through each sheet in the worksheets collection using a For Each (http://excelmacromastery.com/vba-for-loop/#The_VBA_For_Each_Loop) Loop or a For (http://excelmacromastery.com/vba-for-loop/#The_VBA_For_Loop) Loop.

The following example uses a **For Each** loop.

```
Public Sub LoopForEach()

    ' Writes "Hello World" into cell A1 for each worksheet
    Dim sht As Worksheet
    For Each sht In ThisWorkbook.Worksheets
         sht.Range("A1") = "Hello World"
    Next sht

End Sub
```

The next example uses the standard **For** loop

```
Public Sub LoopFor()

    ' Writes "Hello World" into cell A1 for each worksheet
    Dim i As Long
    For i = 1 To ThisWorkbook.Worksheets.Count
         ThisWorkbook.Worksheets(i).Range("A1") = "Hello World"
    Next sht

End Sub
```

You have seen how to access all open workbooks and how to access all worksheets in ThisWorkbook. Lets take it one step further. Lets access all worksheets in all open workbooks.

Note: If you use code like this to write to worksheets then back everything up first as you could end up writing the incorrect data to all the sheets.

```
Public Sub AllSheetNames()

    ' Prints the workbook and sheet names for
    ' all sheets in open workbooks
    Dim wrk As Workbook
    Dim sht As Worksheet
    For Each wrk In Workbooks
        For Each sht In wrk.Worksheets
            Debug.Print wrk.Name + ":" + sht.Name
        Next sht
    Next wrk

End Sub
```

Using the Sheets Collection

The workbook has another collection similar to Worksheets called **Sheets**. This causes confusion at times among users. To explain this first you need to know about a sheet type that is a chart.

It is possible in Excel to have a sheet that is a chart. To do this

1. Create a chart on any sheet.
2. Right click on the chart and select Move.
3. Select the first option which is "New Sheet" and click Ok.

Now you have a workbook with sheets of type worksheet and one of type chart.

- The **Worksheets** collection refers to all worksheets in a workbook. It does not include sheets of type chart.
- The **Sheets** collection refers to all sheets belonging to a workbook including sheets of type chart.

There are two code examples below. The first goes through all the Sheets in a workbook and prints the name of the sheet and type of sheet it is. The second example does the same with the Worksheets collection.

To try out these examples you should add a Chart sheet to your workbook first so you will see the difference.

```
Public Sub CollSheets()

    Dim sht As Variant
    ' Display the name and type of each sheet
    For Each sht In ThisWorkbook.Sheets
        Debug.Print sht.Name & " is type " & TypeName(sht)
    Next sht

End Sub

Public Sub CollWorkSheets()

    Dim sht As Variant
    ' Display the name and type of each sheet
    For Each sht In ThisWorkbook.Worksheets
        Debug.Print sht.Name & " is type " & TypeName(sht)
    Next sht

End Sub
```

If do not have chart sheets then using the **Sheets** collection is the same as using the **Worksheets** collection.

Conclusion

This concludes the post on the VBA Worksheet. I hope you found it useful.

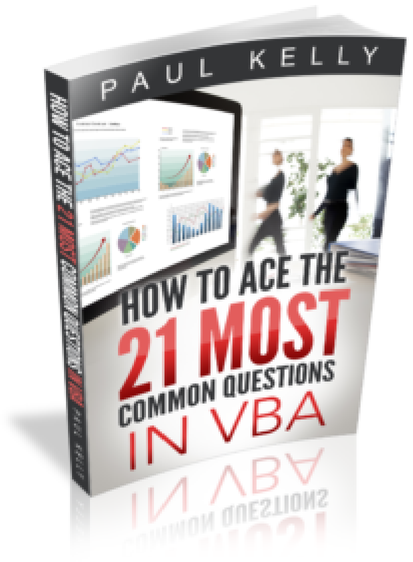
The three most important elements of Excel VBA are Workbooks (http://excelmacromastery.com/excel-vba-workbook), Worksheets and Ranges and Cells (http://excelmacromastery.com/excel-vba-range-cells/). These elements will be used in almost everything you do. Understanding them will make you life much easier and make learning VBA much simpler.

What's Next?

If you want to read about more VBA topics you can view a **complete list of my 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.

If you are planning to build or maintain a real world VBA Application then you may want to check out Build 11 Full VBA Applications (http://www.theexcelvbahandbook.com)

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

Sheets (https://excelmacromastery.com/tag/sheets/) VBA (https://excelmacromastery.com/tag/vba/) Workbooks (https://excelmacromastery.com/tag/workbooks/) Worksheets (https://excelmacromastery.com/tag/worksheets/)

Previous
The Complete Guide To The VBA Workbook (https://excelmacromastery.com/excel-vba-workbook/)

Next
The Complete Guide to Ranges and Cells in Excel VBA (https://excelmacromastery.com/excel-vba-range-cells/)

14 COMMENTS

Peter
September 15, 2015 at 8:30 am (https://excelmacromastery.com/excel-vba-worksheet/#comment-89)

Thank you for the excellent information for a VBA novice of some 10 years

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=89#respond)

Paul Kelly
September 15, 2015 at 9:31 am (https://excelmacromastery.com/excel-vba-worksheet/#comment-90)

You're welcome Peter.

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=90#respond)

George Cook
March 12, 2016 at 10:47 pm (https://excelmacromastery.com/excel-vba-worksheet/#comment-91)

Hi Paul

I've just come across your blog and it looks most informative. I've not had time to read through it yet and the answer to my question may be in there somewhere but in the meantime Is there a shorter way (in a macro) to write If Cell.Value = "A" Or Cell.Value = "B" Or Cell.Value = "C" Then Cell.Value = 2

Many thanks

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=91#respond)

Paul Kelly
March 13, 2016 at 11:32 am (https://excelmacromastery.com/excel-vba-worksheet/#comment-92)

Hi George,

You can use the Select Case statement to do it

```
Select Case Cells(1,1).Value
    Case "A" To "C"
         Cells(1,1).Value = 2
End Select
```

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=92#respond)

George Cook
March 13, 2016 at 10:13 pm (https://excelmacromastery.com/excel-vba-worksheet/#comment-93)

Thanks for that Paul.

In the time between my query and your answer I did find a way except a bit longer than what you have suggested.

This is what I found

```
For Each oCell In Range(“B11:M32”)
Select Case oCell.Value
Case Is = “A”, “B”, “C”: oCell.Value = 2
Case Is = “D”, “E”, “F”: oCell.Value = 3
Case Is = “G”, “H”, “I”: oCell.Value = 4
Case Is = “J”, “K”, “L”: oCell.Value = 5
Case Is = “M”, “N”, “O”: oCell.Value = 6
Case Is = “P”, “Q”, “R”, “S”: oCell.Value = 7
Case Is = “T”, “U”, “V”: oCell.Value = 8
Case Is = “W”, “X”, “Y”, “Z”: oCell.Value = 9
End Select
Next
```

Anyway thanks again. Much appreciated.

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=93#respond)

mike denley
August 9, 2016 at 7:04 pm (https://excelmacromastery.com/excel-vba-worksheet/#comment-94)

I use the code name to identify the different types of worksheets that are being created/used; allowing users to rename the worksheet as they see fit. From your experience what is the best way to set the code name ?

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=94#respond)

Paul Kelly
August 10, 2016 at 8:25 am (https://excelmacromastery.com/excel-vba-worksheet/#comment-95)

Hi Mike,

I normally set it manually in the property windows. It can be done using the code but normally this is not necessary.

Paul

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=95#respond)

harsha547

April 4, 2017 at 4:08 pm (https://excelmacromastery.com/excel-vba-worksheet/#comment-4891)

Thanks for sharing !!

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=4891#respond)

Jorge C

April 13, 2017 at 3:10 am (https://excelmacromastery.com/excel-vba-worksheet/#comment-5341)

Hi Paul,
I am really really really thankful for your very informative, didactic and amusing posts. I have learned a lot since I began to read and rewrite your code, while adding slight changes and experimenting.
I just have a comment, I found that into the Sub AllSheetNames(), the second “next” asks for an “i” variable where the proper one should be “wrk”
Thank you again for everything and keep making live videos.

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=5341#respond)

Paul Kelly

April 13, 2017 at 3:54 am (https://excelmacromastery.com/excel-vba-worksheet/#comment-5345)

Thanks Jorge, I have update the code. Glad you enjoy the website posts.

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=5345#respond)

Afzal khan

April 26, 2017 at 1:12 pm (https://excelmacromastery.com/excel-vba-worksheet/#comment-5764)

Thanks for your great effort Paul Kelly.
It's very helpful to developing towards VBA programmer.

But one question, you have not shown how the "codename" for the worksheet is made. Only "codename" is displayed in the property screen shot.

Thanks for sharing knowledge…. I am glad i found this site.

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=5764#respond)

Afzal khan

April 26, 2017 at 1:15 pm (https://excelmacromastery.com/excel-vba-worksheet/#comment-5765)

Ya i got it, its already mentioned there.
Sorry i missed it.

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=5765#respond)

Paul Kelly

April 27, 2017 at 3:34 am (https://excelmacromastery.com/excel-vba-worksheet/#comment-5782)

No problem Afzal.

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=5782#respond)

Curzio

June 19, 2017 at 8:04 am (https://excelmacromastery.com/excel-vba-worksheet/#comment-8105)

Hallo Paul, your whole site is the mandatory start lane and pit stop for every vba writer.
I get confused by example "Code Name in other Workbooks": I miss somewhere something like

Dim iWB As Workbook
For Each iWB In Application.Workbooks
‘Loop thru Worksheets and check condition
Next

Thank again and kind regards

Reply (https://excelmacromastery.com/excel-vba-worksheet/?replytocom=8105#respond)

LEAVE A REPLY

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

Name *

Email *

Website

Post Comment