Stop flickering / refactor code for Excel ScreenUpdating false on Copy Paste to another worksheet

Asked 5 years, 9 months ago Modified 5 years, 9 months ago

Viewed 407 times

4

I am beginner and still learn about programming MS Excel VBA macros. I need help from the community to solve my problem with macro code on excel.

```
    Sub export_data()

With Application
    .ScreenUpdating = False
    .Calculation = xlManual 'sometimes excel
calculates values before saving files
End With

Dim wsCopy As Worksheet
Dim wsDest As Worksheet
Dim wsDest2 As Worksheet
Dim lCopyLastRow As Long
Dim lDestLastRow As Long
Dim lDestLastRow2 As Long
Dim i As Long
Dim check As Long

  'Set variables for copy and destination sheets
  Set wsCopy = Workbooks("Book
1.xlsm").Worksheets("Sheet 1")
  Set wsDest = Workbooks("Book
2.xls").Worksheets("Sheet 1")
  Set wsDest2 = Workbooks("Book
2.xls").Worksheets("Sheet 2")
```

```
  '1. Find last used row in the copy range based
on data in column A
  lCopyLastRow =
wsCopy.Range("J10:J16").Find(what:="",
LookIn:=xlValues).Offset(-1).Row

  '2. Find first blank row in the destination
range based on data in column A
  'Offset property moves down 1 row
  lDestLastRow = wsDest.Cells(wsDest.Rows.Count,
"J").End(xlUp).Offset(1).Row
  lDestLastRow2 =
```

I'm using Microsoft Office 2016. When I run the code it is running well but still flickering. It's disturbing and I'm afraid it will slow processing.

Any idea to stop the flickering when code is running?

excel

Share

Improve this question

Follow

asked Mar 21, 2019 at 10:04

Poetoe

41 ● 2

2 Answers

Sorted by: Highest score (default)

4

The simplest things first:

If you're going to be doing and VBA development look into Rubberduckvba.com It's an add-in that will make coding much easier and teach you a whole lot you didn't

know you wish you knew. Full disclosure I'm a contributing member to that group.

`Option Explicit` isn't shown in your code. Also because you have an undeclared variable `cell` in your export code I'm assuming you don't have that turned on by default. Under the menu at the top Tools>Options>Editor tab>Code Settings group>Require Variable Declaration check that box. This mandates you have `Dim cell As Range` before you can use a variable. With the option turned on you'll get a Compile error of **Variable not defined** before you can run your code. This may seem like something minor but turn this option on as it will save you headache later.

You're using `check` as a message box result. Don't declare it as a `Long`, rather declare it `Dim check As VbMsgBoxResult` that way when you type `check=` you'll get intellisense and the enumeration values available to you.

You have `""` used as a placeholder for an empty string. Use `vbNullString` instead. It's a built-in constant that lets you know this check was intentional. This is because `""` *might* or *could* have been a string that had a value, `"CheckValue"`, that had the word removed leaving only the empty quotes. `vbNullString` is unambiguous.

I left most of your variable names along so you could more easily follow along with the refactoring I did. Note that Variables like r, x, xMax, don't provide any useful information about what they're used for. Use descriptive variable names. Future you will thank you. Descriptive

variables make code self documenting and much easier to read.

Comments. Comments can be a hot topic for some people. I've found with descriptive variables you need less code. The code itself should say *what* is being done. Your comment "'1. Find last used row ..." is saying exactly what it's doing again. `lastRowInCopyArea = copyWorksheet.Range().FooBar.Row` is already saying that. Save comments for *why* something is done. The *what* should be apparent from the code itself.

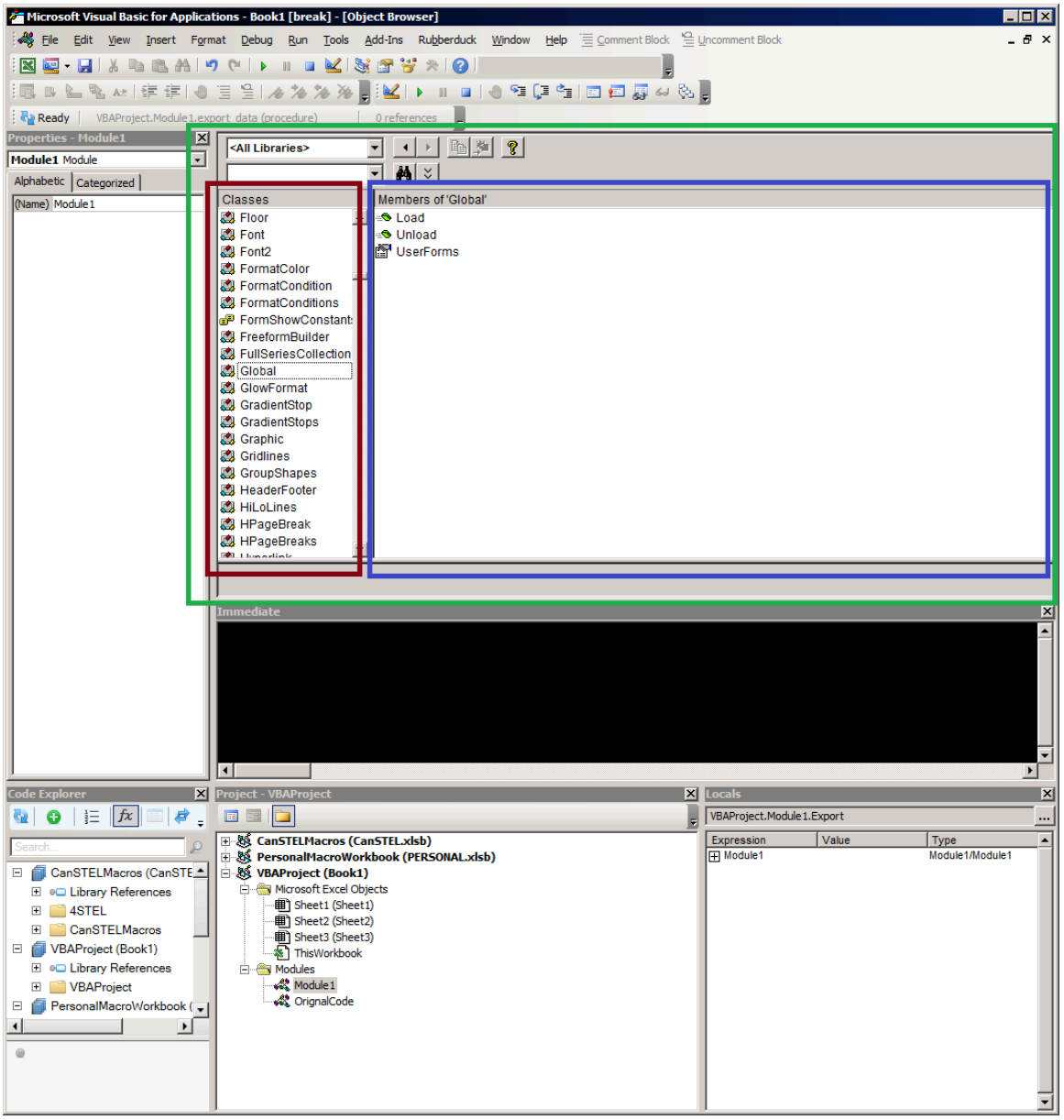
Hungarian Notation (HN) isn't needed. The Integrated Development Environment (IDE) can tell you what the type of the variable is from the menu Edit>Quick Info Ctrl+I. Having a letter indicating a type inhibits readability and is a carryover from previous coding habits. Good variable names will remediate a lot of this by itself.

You can use a typed `UCase$()` function instead of the generic `UCase()` at the beginning for your export section since you're dealing with strings.

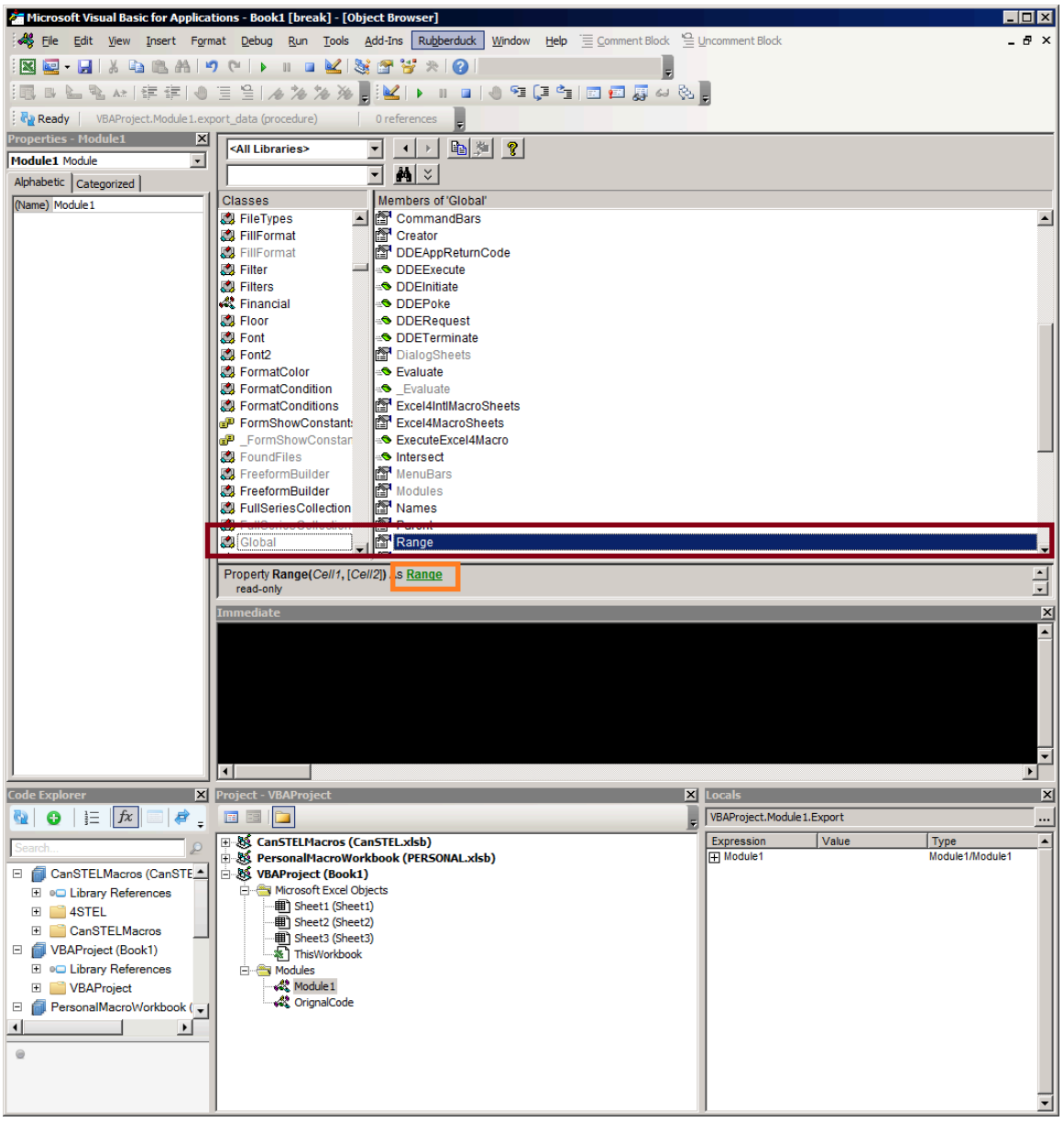
You are using things implicitly. Your `Range(Foo)` are implicitly accessing the active sheet you're on. To see this right click on the word Range to bring up the context menu and select **Definition**.

When you do this you'll probably get a dialog stating "Cannot jump to 'Range' because it is hidden", under which the Object Browser is now displayed (Green).

Dismiss the dialog by clicking OK. Right click within either the Classes (Red) or Members (Blue) pane area and choose **Show Hidden Members** from the context menu.



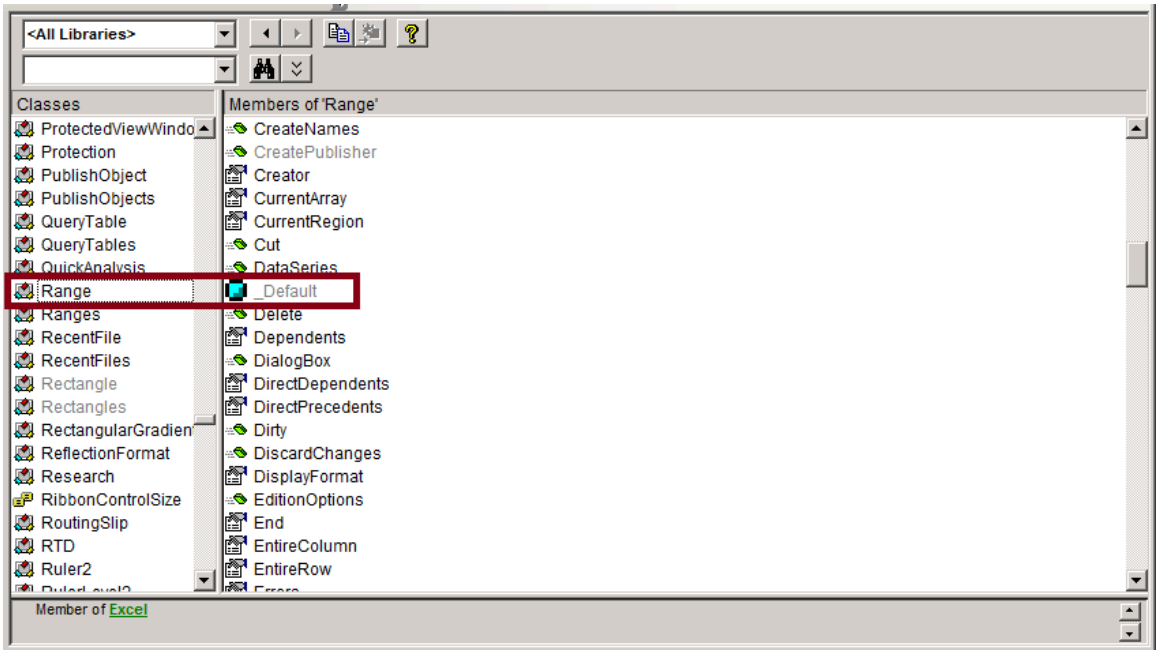
Close the Object browser by clicking the inner close button in the top right corner or use Ctrl+F4. Your code window will now be displayed. Again bring up the context menu by right clicking on the word Range and choose Show Definition. You're taken to the hidden Global class and the Range member.



The red box shows the greyed class name `Global` is normally hidden and the `Range` member is what's accessed. To avoid this implicit access fully qualify your Range with a worksheet or `ActiveSheet.Range(Foo)` if you do want to access the active sheet. Again doing this is unambiguous and shows it's intentional.

We've got the left side of `Range(Foo)`, now what about the other side? You're also implicitly accessing the default property. How do you figure this out? In the image above, within the orange box, the word Range is green indicating it's a link. Click on it and you'll be taken to Range in the

Classes pane, shown below. A Range object has members that can be accessed, either Methods (things that do an action) or Properties (information about the range).



The Members pane shows these members you can access. Scroll down in the Members pane until the `_Default` member is shown. When you don't include a member access IE `Range(Foo)` you're accessing the `_Default` member. Since you're checking the value of the cell use `Range(Foo).Value2` to qualify your member access.

Your looping can and should be consolidated. Take the first loop and compare it with the others. Any time you copy/paste and add a number identifier to a variable you have a code smell. The start row is 10 for each of them, only the column varies.

```
    Dim r As Range, tabel As Range, xTabel As
Range
    Dim x As Integer, xMax As Long
    'y As Long, yMax As Long
    Dim textTabel As String
    Set tabel = wsCopy.Range("d10:d" &
lCopyLastRow)
    Set r = wsDest2.Range("d" & lDestLastRow2)

    xMax = tabel.Rows.Count
    For x = 1 To xMax
        Set xTabel = tabel.Range(Cells(x, 1),
Cells(x, 1))
        textTabel = Trim(xTabel.Text)
        If x = 1 Then
            textTabel = textTabel
            'r.Offset(x - 1, 0).ClearContents
        Else
            textTabel = "& " & textTabel
        End If
        r = r & textTabel
     Next x
```

You need to pull this into its own Function that describes what it's doing. Doing this will eliminate duplicate code. Another benefit of this is if you catch a bug and you fix it anywhere that calls/uses the function will also be fixed.

What is your code doing? It's concatenating cells in a range to make a text label. Let's start start with that for the name `ConcatenateLabelFrom`. I saw your variable `r` is assigned every time within the loop. You don't need to do this, only once all the concatenation is done. Remember that this will be the range that's used for the destination. The logic of the loop can be condensed down to

```
Private Function ConcatenateLabelFrom(ByVal
concatenateArea As Range) As String
    Dim rowInArea As Integer
    For rowInArea = 1 To
concatenateArea.Rows.Count
        Dim textTabel As String
        textTabel =
Trim(concatenateArea.Cells(rowInArea).Text)
        If rowInArea = 1 Then
            textTabel = textTabel
        Else
            textTabel = textTabel & "& " &
textTabel
        End If
    Next

    ConcatenateLabelFrom = textTabel
End Function
```

The function is called by supplying it with an argument to the parameter as follows. The indentation is only there for ease of readability.

```
    wsDest2.Cells(lDestLastRow2, "d").Value2 =
ConcatenateLabelFrom( _

wsCopy.Range( _

wsCopy.Cells(10, "d"), _

wsCopy.Cells(lCopyLastRow, "d") _

) _
                                                )
```

Your jumps with GoTo aren't needed. Better that you restructure your code than jumping around with GoTo.

Doing this will have you code flow more logically. It'll also require you to think about how you want to restore the `Application.ScreenUpdating/Calculation` properties.

You could do this by encapsulating the sections into in their own subs. Your Protect sub would be as follows and called via `Protect wsCopy, protectBook`. And similar could be done with the Export.

```
Private Sub Protect(ByVal worksheetToProtect As
Worksheet, ByVal workbookToSave As Workbook)
    worksheetToProtect.Protect "pass", _
                AllowFormattingCells:=True, _
                DrawingObjects:=True, _
                contents:=True, _
                Scenarios:=True
    workbookToSave.Save
End Sub
```

Your section that has

Your screen flickering looks to be occurring because you restore screen updating and automatic calculation before Export. You have copy and pasting going on there and that's what's being shown. Remember my comment about `r` being assigned within the loop? That's part of this. You can use Application.Calculate to calculate all open workbooks before turning back on ScreenUpdating. As with refactoring your GoTo jumps, think through how you want your workbooks series of events to happen and code them accordingly.

There's more that can be suggested but this should suffice for a start.

Share Improve this answer

Follow

answered Mar 21, 2019 at 22:06

IvenBach

573 ● 5 ● 13

2

Actually, this is not a very good practice to use the GoTo statement in VBA, you'd better **split your code in several functions** (or even modules) to make the whole code more readable.

You can then use if/then/else of select/case statements to handle every part. Flickering is probably bound to the fact that you reactivate `ScreenUpdating` *before* some part of the code execution.

This block of code:

```
With Application
    .ScreenUpdating = True
    .Calculation = xlCalculationAutomatic
End With
```

Should be run at the very end.

Share Improve this answer

Follow

answered Mar 21, 2019 at 14:57

JMax

26.6k ● 12 ● 73 ● 89