Developer Tab Part 2. - Creating Macros to simplify repetitive tasks in Excel

Macros are defined by Dictionary.com as: an instruction that represents a sequence of instructions in abbreviated form. Basically a Macro represents a series of commands that are stored in VB (Visual Basic) code with an Excel Spreadsheet. These commands may be called upon, and when executed, the Macro code will repeat the commands on a given spreadsheet.

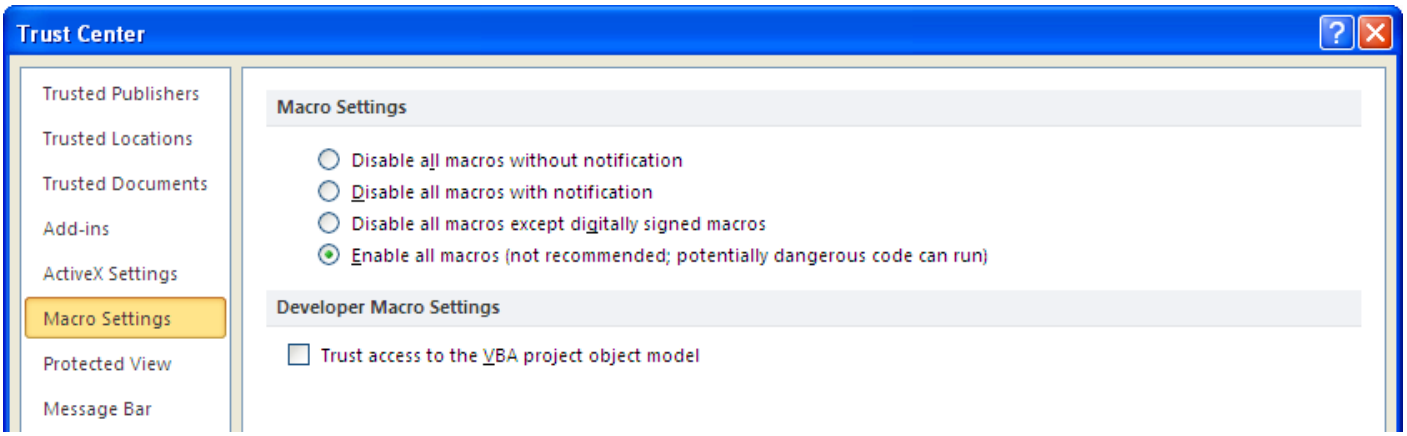
Macros are very powerful tools, and can be used to automate a number of tasks within excel. NOTE: Macros are code. When code is run, it runs exactly as it is told to run. If there are errors in the code, it may return undesired results. As one client told me, "These Macros are really great. The only thing is they do what you TELL them to do, not what you THINK they should do."

Enabling Macros in Excel

By Default, Excel has Macro security set to prompt the user that there are macros contained within the workbook. You may choose to leave the security settings in the default mode which will prompt for activation / use of the macro, or you may disable the warnings and allow all macros to run.

Excel 2007 – Office Button – Excel Options – Trust Center – Trust Center Settings – Macro Settings

Excel 2010 – File – Options – Trust Center – Trust Center Settings – Macro Settings



Here we see the options for using Macros in Excel. By Default, the setting is "Disable all macros with notification". This selection will prompt the user with a *Microsoft Security Warning* that there is code enabled in the workbook that contains a macro. It is up to the user what setting is used. If you are confident that you are not going to download random Excel Sheets from the internet, you can probably feel safe to use the "Enable all macros" choice.

From the Microsoft web site :

Trust access to the VBA project object model : This setting is for developers and is used to deliberately lock out or allow programmatic access to the VBA object model from any Automation client. In other words, it provides a security option for code that is written to automate an Office program and programmatically manipulate the Microsoft Visual Basic for Applications (VBA) environment and object model. This is a per user and per application setting, and denies access by default. This security option makes it more difficult for unauthorized programs to build "self-replicating" code that can harm end-user systems. For any Automation client to be able to access the VBA object model programmatically, the user running the code must explicitly grant access. To turn on access, select the check box.

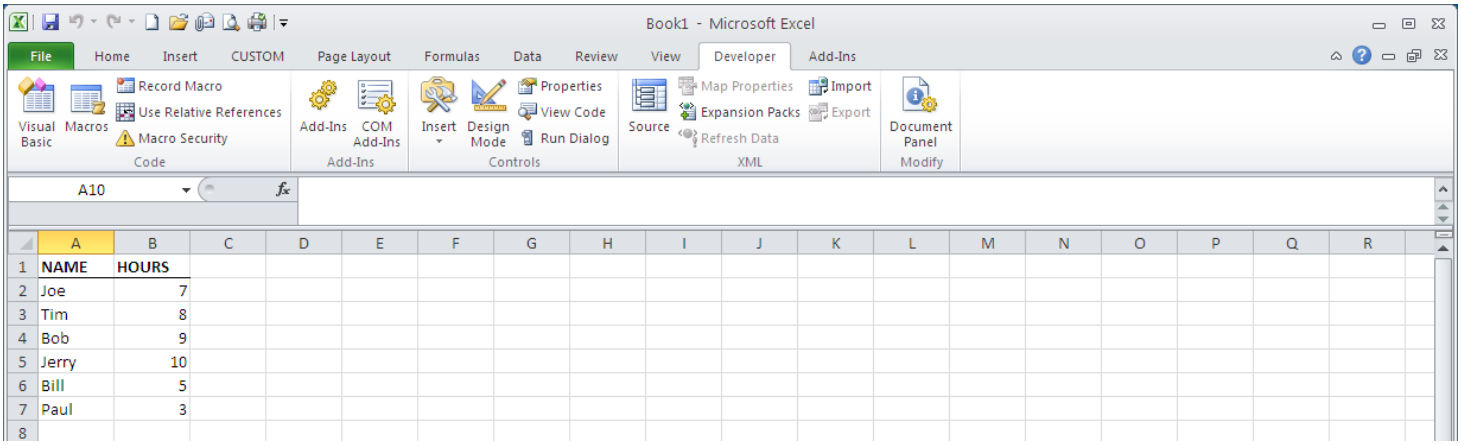
Recording a Macro within a spreadsheet

This example will show a very simple macro to:

- Copy the Contents of an Existing Worksheet
- Create a New Worksheet
- Paste the Contents of an Existing Worksheet into the New Worksheet
- Format the new values within the new Worksheet

Access the Developer Tab in Excel. In the upper left hand corner, you will see the CODE section that contains the Macro options.

The task is simple, we want to create a macro that will copy the information from the existing sheet, create a new sheet, paste the information into the new sheet and remove the values from the hours column.

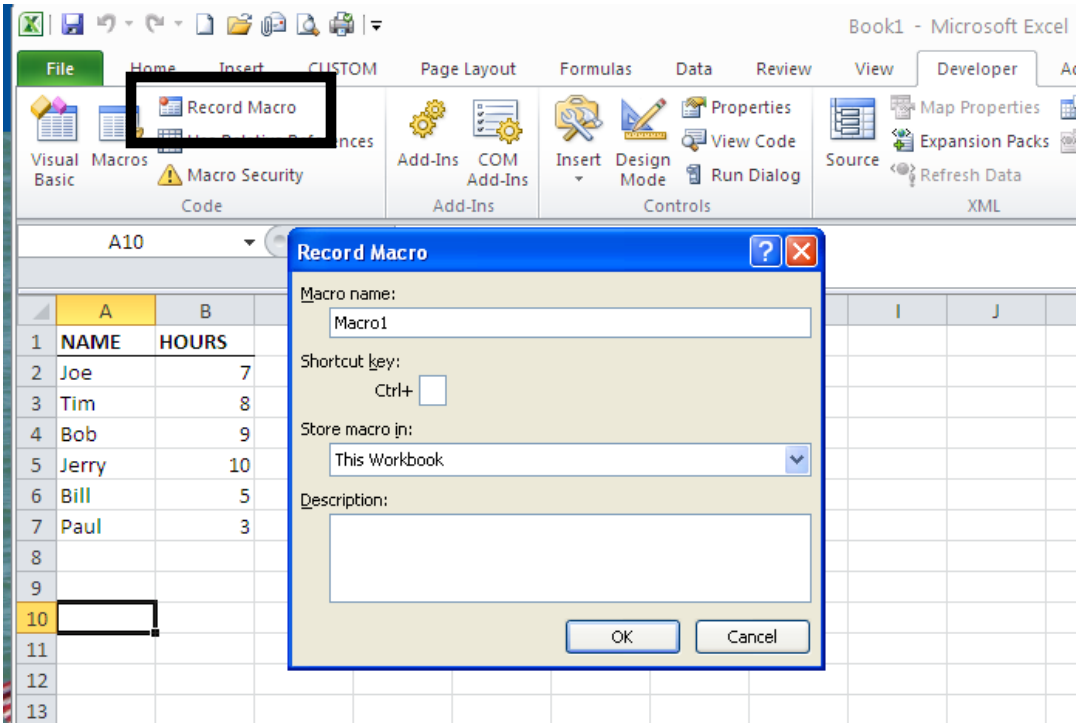


To create the macro, we will use the **Record Macro** function. NOTE : you are "LIVE" any and all keystrokes that you perform while recording a macro will be captured. If you make a mistake while recording a macro, it will record the mistake. When you play the macro back, it will repeat the mistake. This may return undesired results. Take care to Plan the Work and Work the Plan before attempting to record macros.

Over time, you will develop the ability to look at the code and determine what does and doesn't belong in the VBA code. (More on this later).

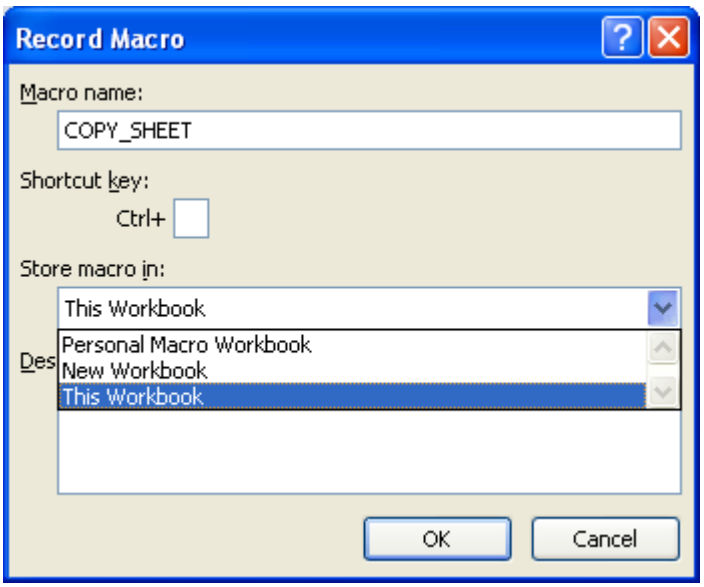
I have defined the tasks I want to complete at the top of the page.

First, select the option to ***Record Macro*** from the Code section of the Developer Tab.



You will be prompted with a dialog box that prompts you to name the macro. The default name is Macro1. You may change this title, but you may not use special characters or spaces in your Macro Name.

In this example we will call the macro COPY_SHEET. Notice that an underscore is used in lieu of a space.



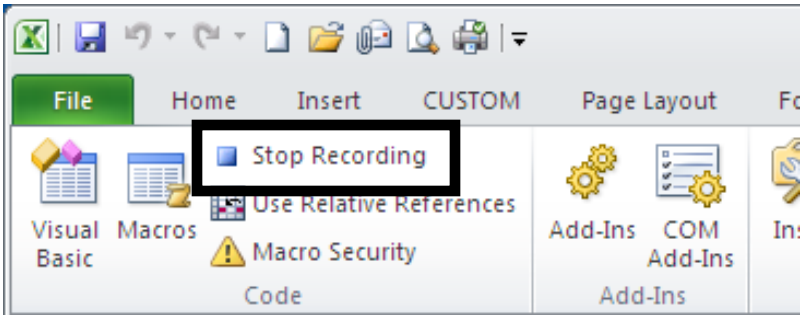
You may also select a shortcut key to execute the macro – something like Ctrl + "C"…. although this is a shortcut that is already reserved for "Copy". You may find it difficult to choose a shortcut key that is already not used.

Store Macro in: In this example, we are storing (or saving) the macro in the current Workbook. If you choose to store the macro in the PERSONAL workbook, the macro will be available in ALL workbooks that you use. This feature is a whole other session that we may cover in a future webinar.

Description: This field is used for a brief description of the macro's use.

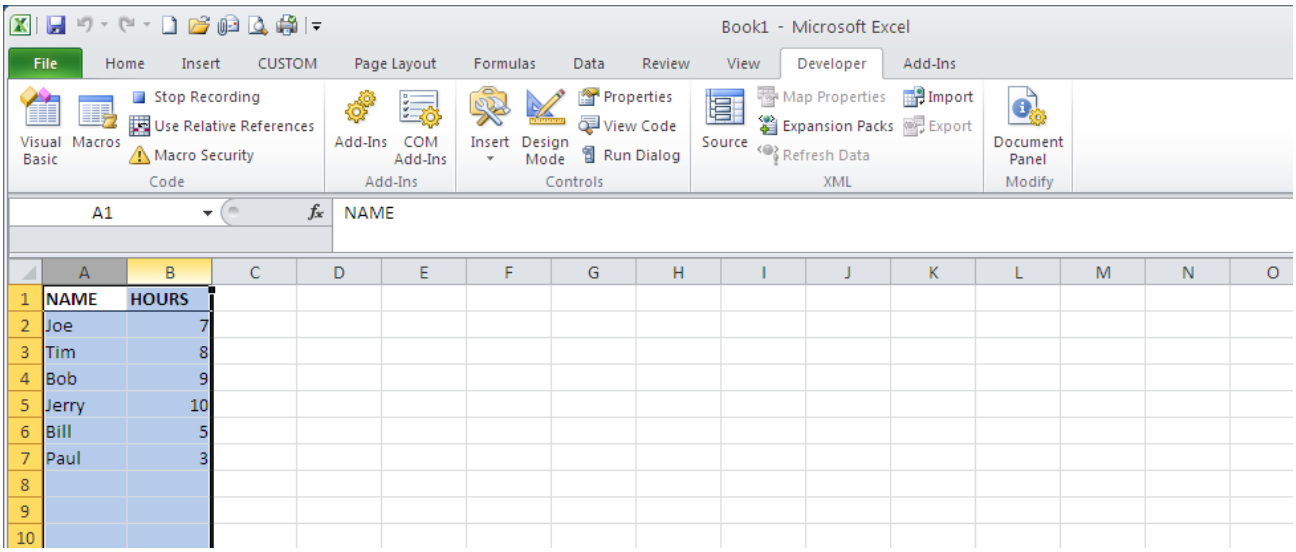
As soon as you click “OK” – you are now recording the Macro. Every click and keystroke is being recorded, so take care to minimize anything moves or clicks that are not part of your task.

The first thing to notice is, when in recording mode, the upper left section of Excel shows “***Stop Recording***” instead of “***Record Macro***”.

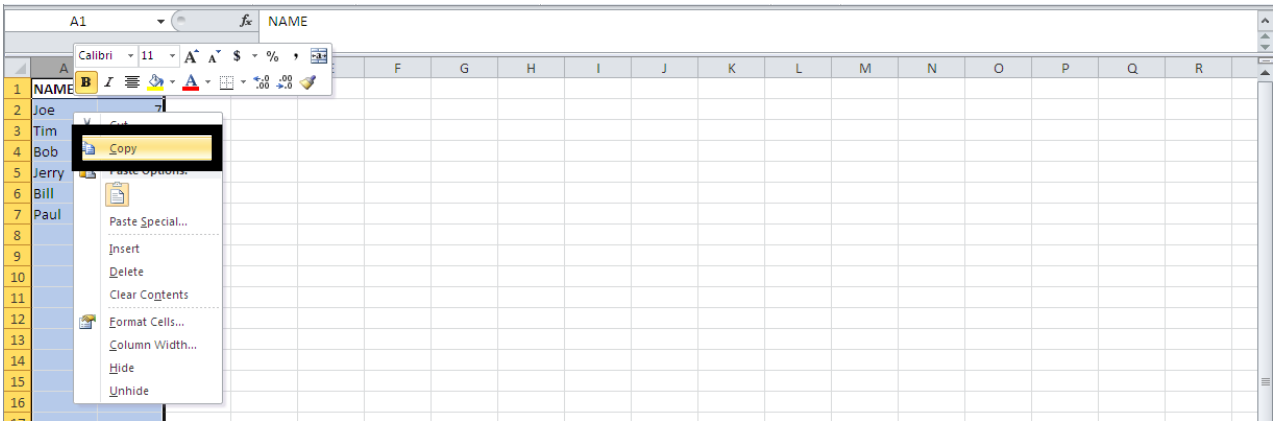


First Step:

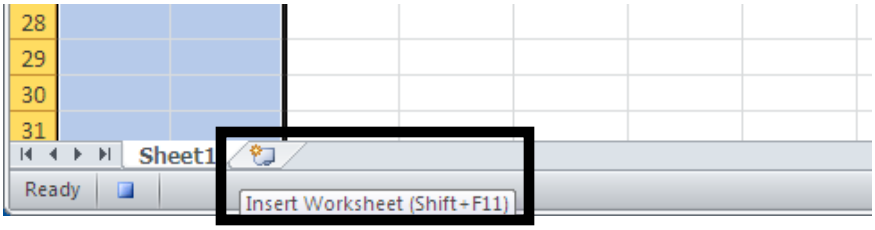
Highlight all of columns A and B in the spreadsheet. We are highlighting the entire column, not just the available data range, as this original sheet may have more names added at a later date. We want the macro to capture any information contained in columns A and B.



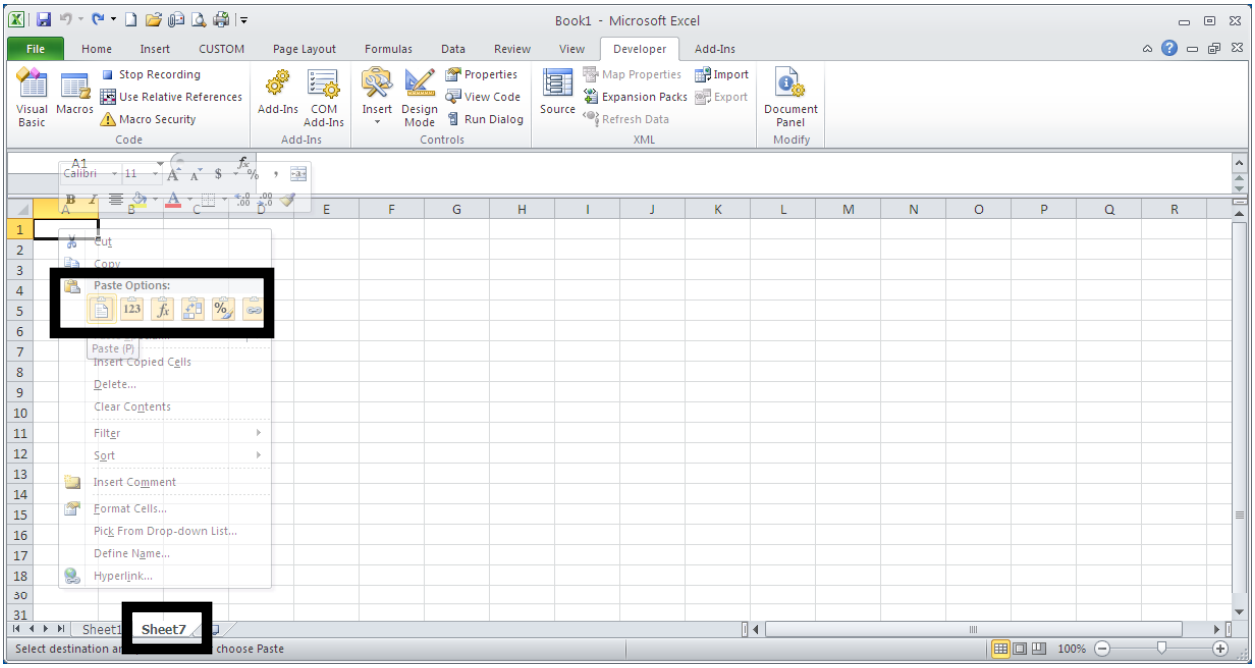
We will then right click on the highlighted data range and select **COPY**. It is important to Right click on the COLUMN HEADER, and NOT within a cell, as we want to copy the entire contents of the column(s).



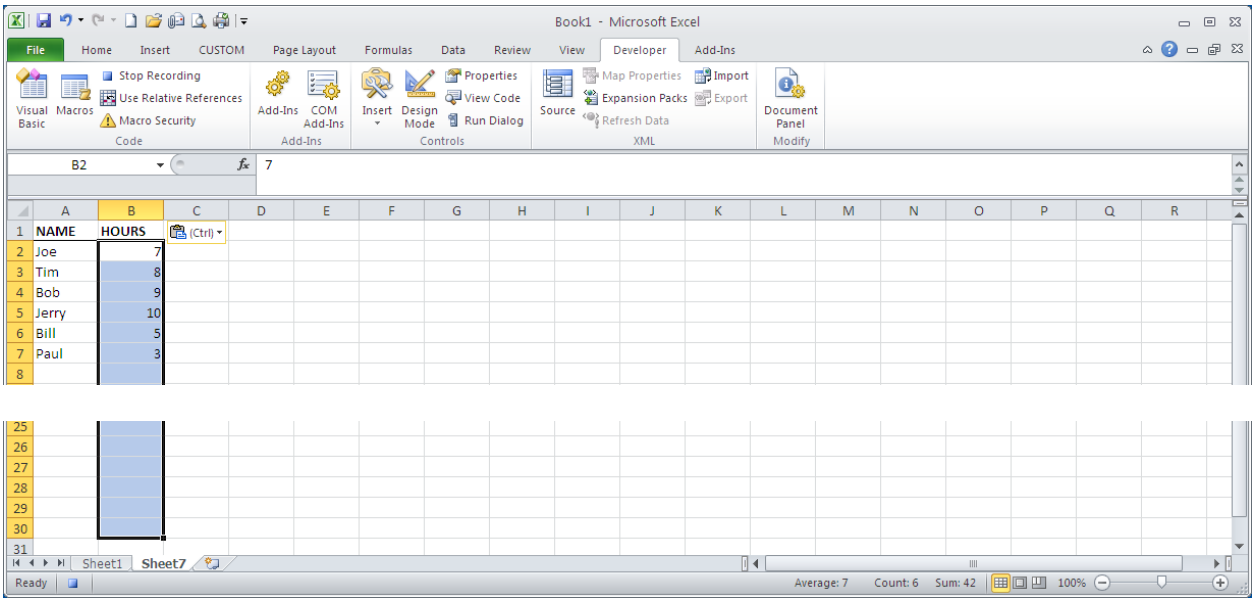
Insert a New Worksheet by clicking the icon in the lower left hand corner of the workbook.



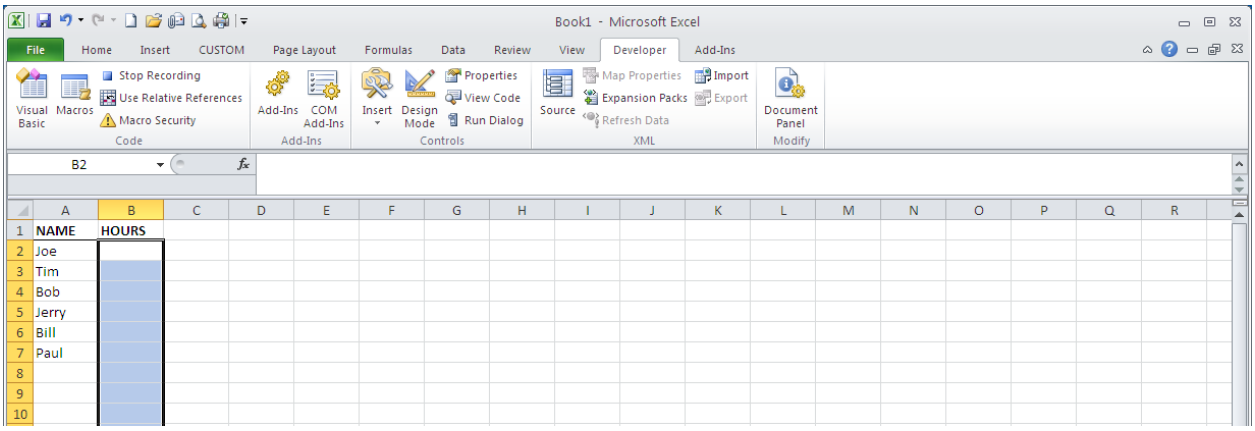
Then, access the new worksheet. Right click on cell A1 and select Paste (P).



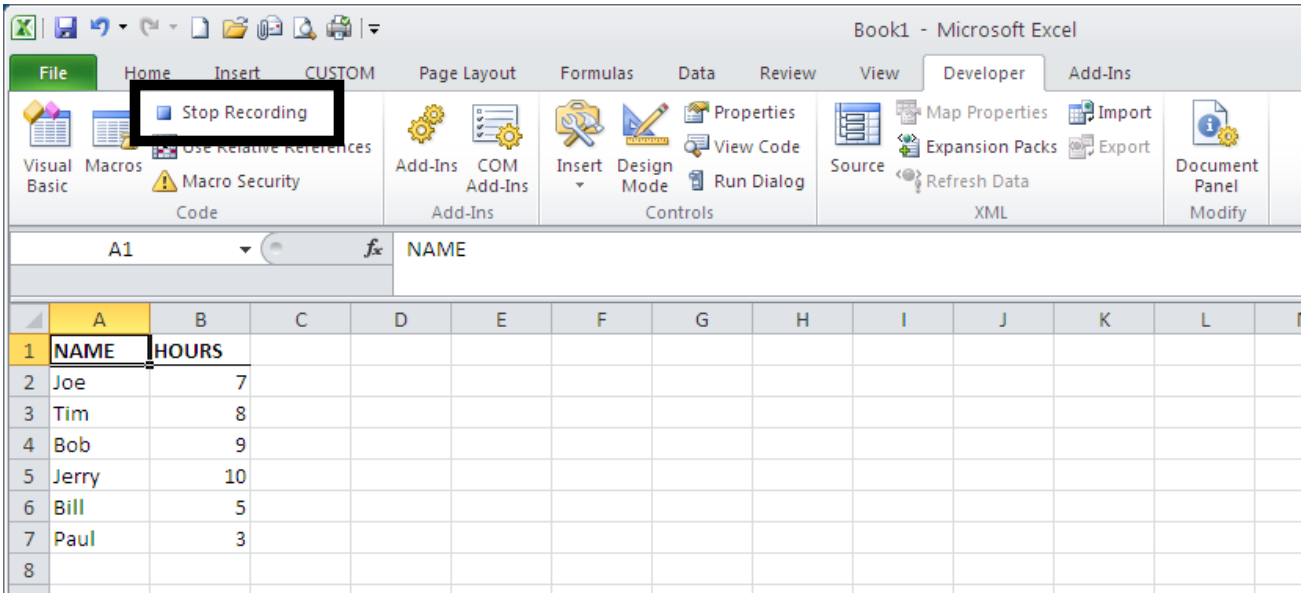
We will not highlight the values in Column B and delete. We do not want to highligh all of column B, only the values for the hours. This will require a decision of "how far" you need to go in order to make this usable for future lists (ie: 30 more employee names added to the original list).



Once the appropriate range is selected, simply press the delete key on the keyboard.



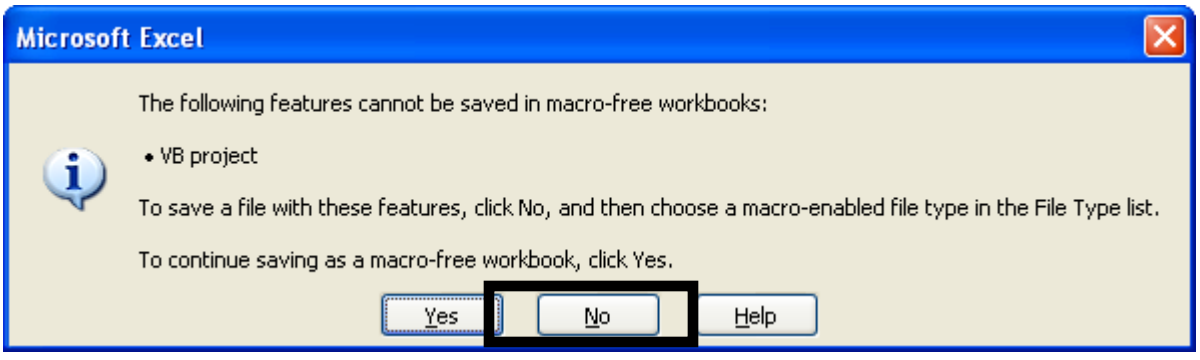
Click on the original sheet (Sheet1) again to finish the macro where it started. Then click the Stop Recording button.



OK – so the macro has been recorded…. Now what?

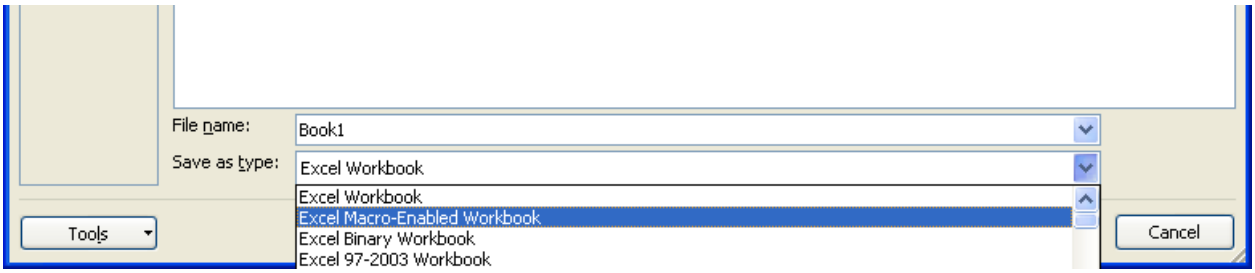
At this point, I like to save the workbook with the macro intact. If I get undesired results from the macro being run, I can always revert to the save point BEFORE the macro has been executed.

If you perform the traditional File – Save, you may receive a warning message.



Basically, newer versions of Excel require you to save as a Macro enable workbook. CLICK NO here ! (If you select "yes", your macro will be lost.)

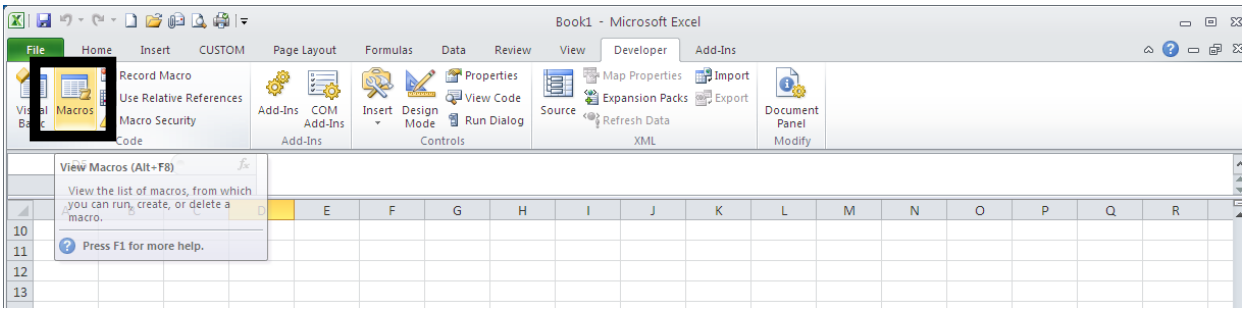
Select the option to Save as tyoe: Excel Macro-Enabled Workbook and click SAVE.



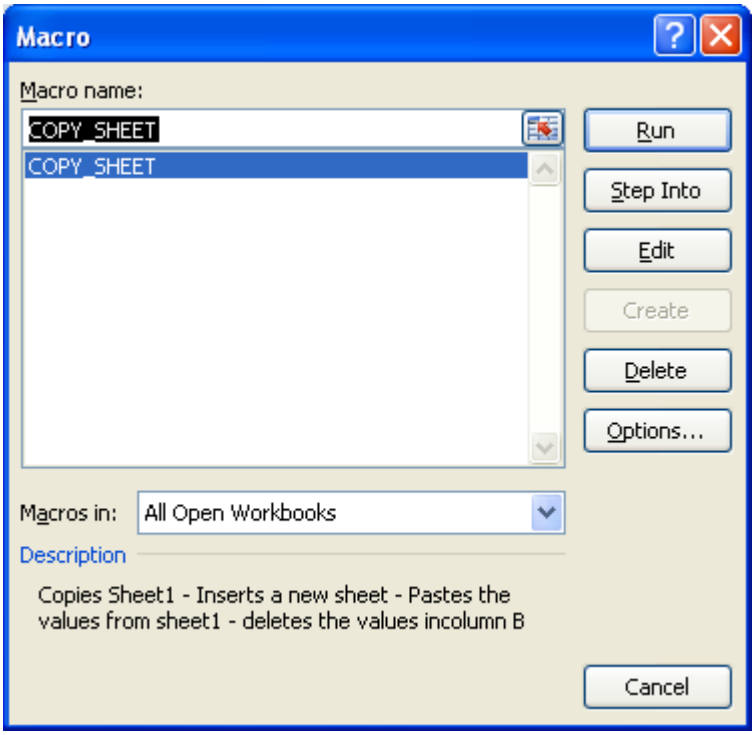
This will save the contents of the workbook along with the VB Code that contains the “program” for the Macro.

Test the Macro

In the Upper Left hand corner of the Developer tab, selec the button for MACROS.

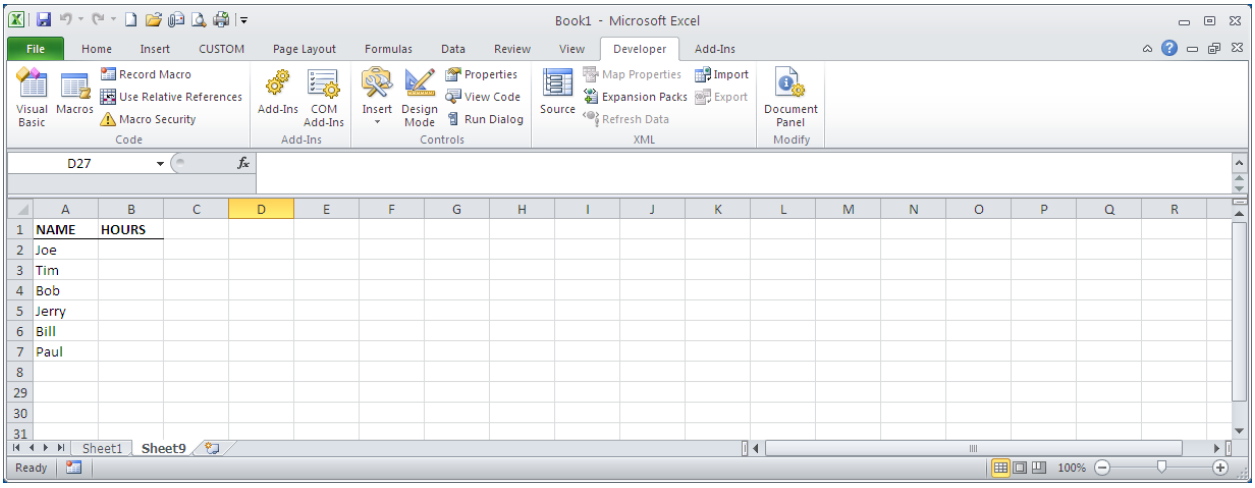


You will be shown a list of existing macros within the OPEN WORKBOOKS. If you have other Macro-Enabled workbooks open, you will see a list of Macros from those workbooks in this window as well.

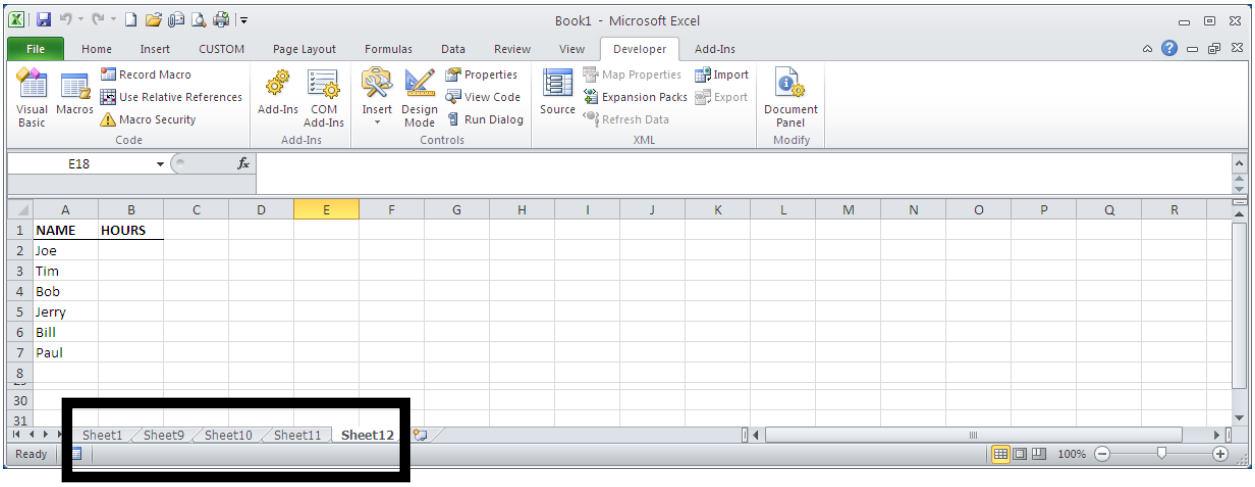


Take a deep breath and Click RUN.

The screen will flicker as the macro is executed. Look at the results. We have a new sheet with the desired results.

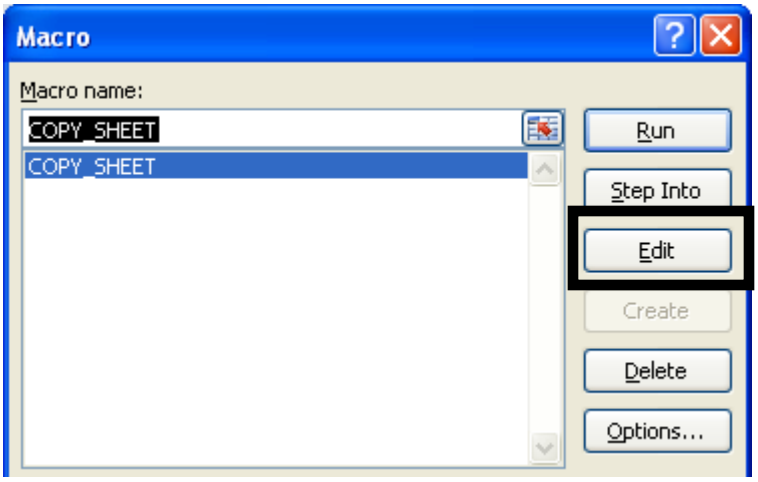


Select Sheet1 and run the macro again. As long as the code is correct, the results will be the same every time.

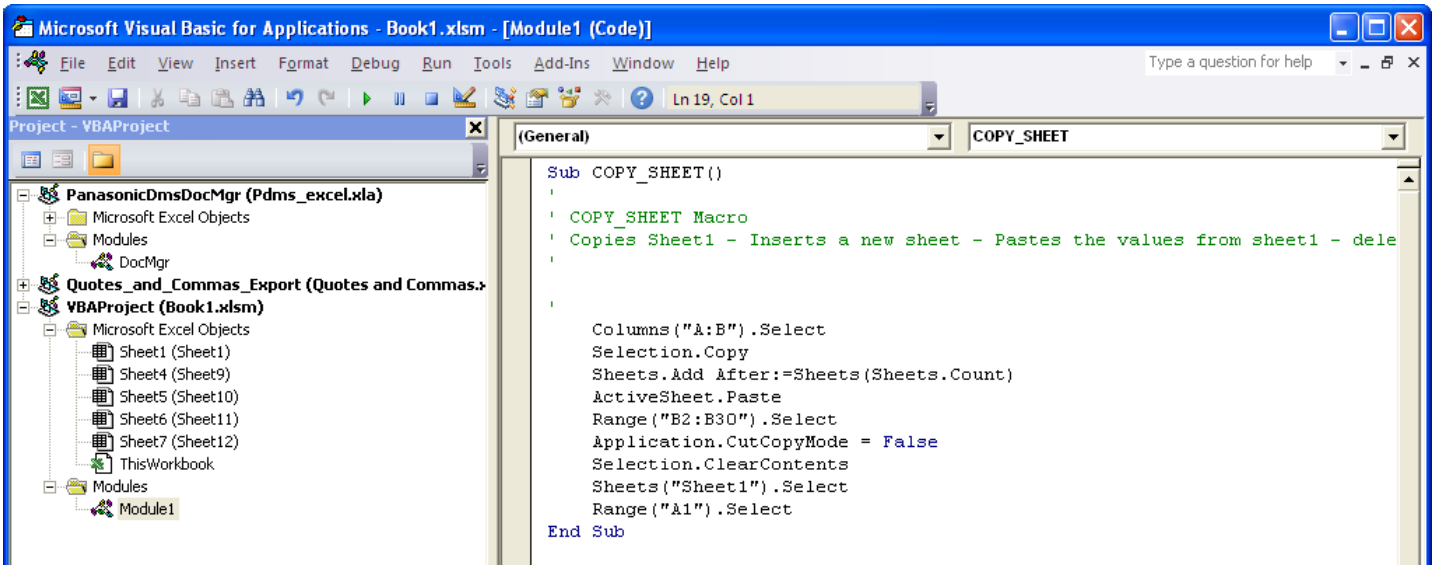


Viewing the VB Code – troubleshooting macros

We could spend an entire year's worth of webinars on VB Code. Here are some tips in helping you understand what we just did. Select the MACRO option from the Developer tab, and choose EDIT.



This will open the VB (Visual Basic) editor. This is the programming code that represents all of the functions that we just recorded.



You will notice that the code is easy to read, but for the novice, it is difficult to write.

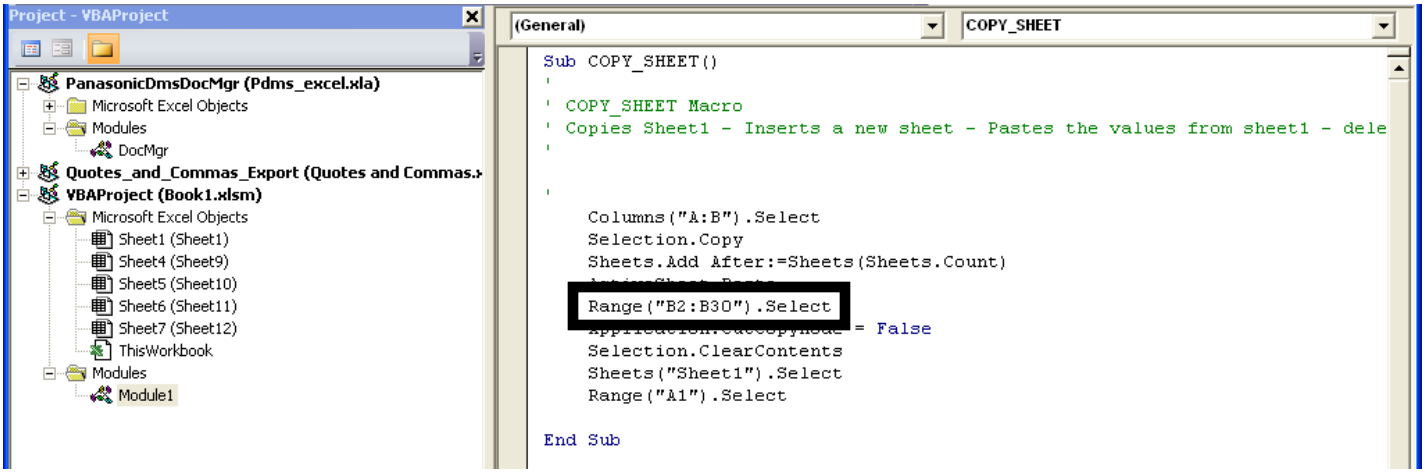
Basically, the code is written to replicate all of the moves we performed while recoding the macro.

Columns("A:B").Select	Selects all values in Columns A and B
Selection.Copy	Copies the selected range
Sheets.Add After:=Sheets(Sheets.Count)	Adds a new worksheet
ActiveSheet.Paste	Performs the Paste Function into the new worksheet
Range("B2:B30").Select	Selects the desired range B2 – B30
Application.CutCopyMode = False	Deactivates the CUT/COPY option because the delete key was pressed.
Selection.ClearContents	Clears the contents of the selected cells
Sheets("Sheet1").Select	Selects Sheet1
Range("A1").Select	Selects Cell A1

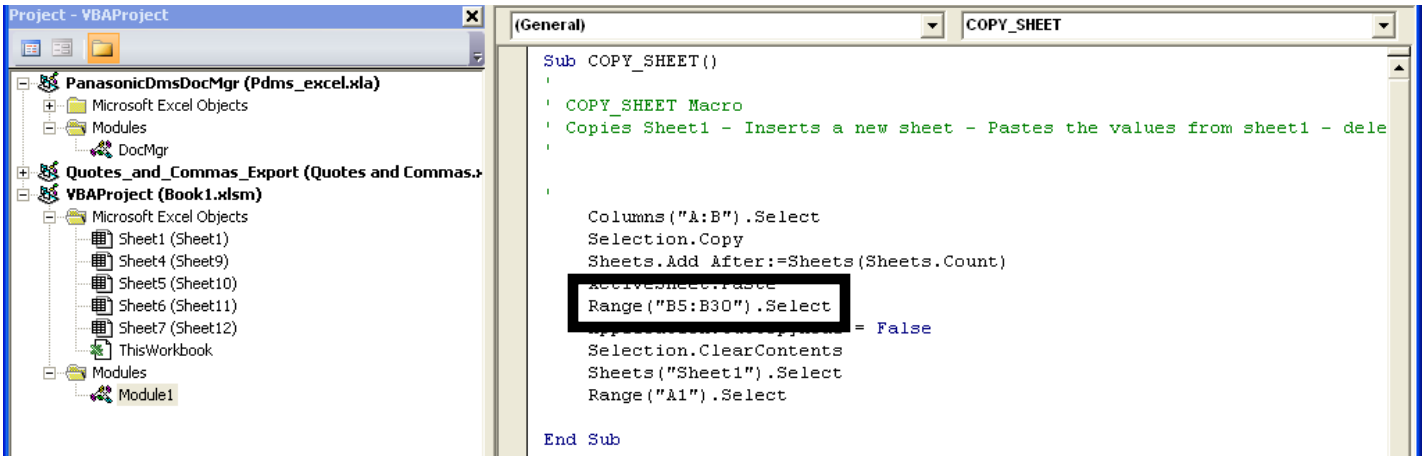
Editing the Code

Just for giggles, let's make a change to the code. Say we want to change the range of cells that we want to delete the hours from once the data has been copied. If we want to maintain the Hour values for the first three employees, we would make the following change(s).

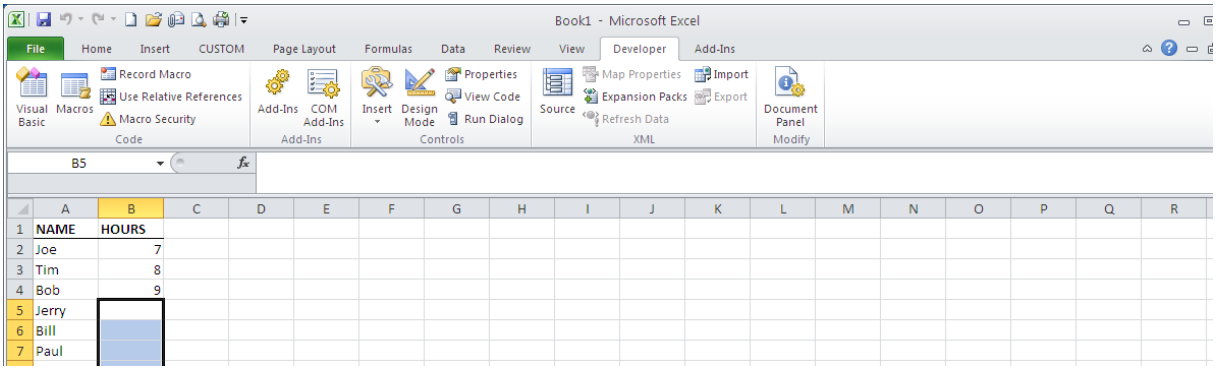
The current code is set to select the values in B2:B30.



We are going to change the code from B2:B30 to B5:B30.



Close the VB editor window and test the macro.

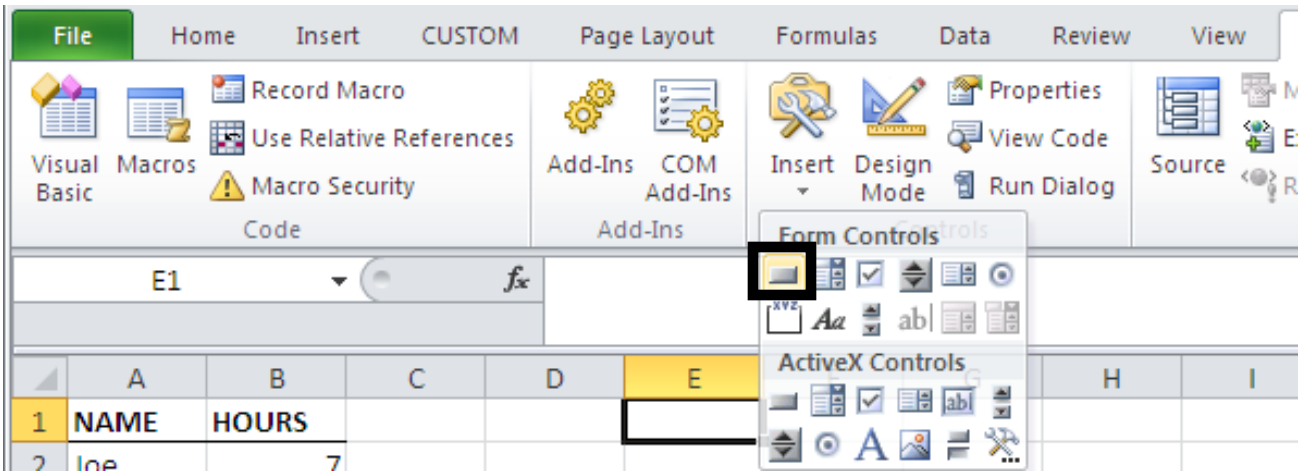


Success. Manually editing macros should be performed with great care, as you may lose functionality with the macro if you make a coding mistake. It is always a good idea to save the spreadsheet before you make changes. You may also copy the Macro VB Code out to a Word Document and Paste it back if need be.

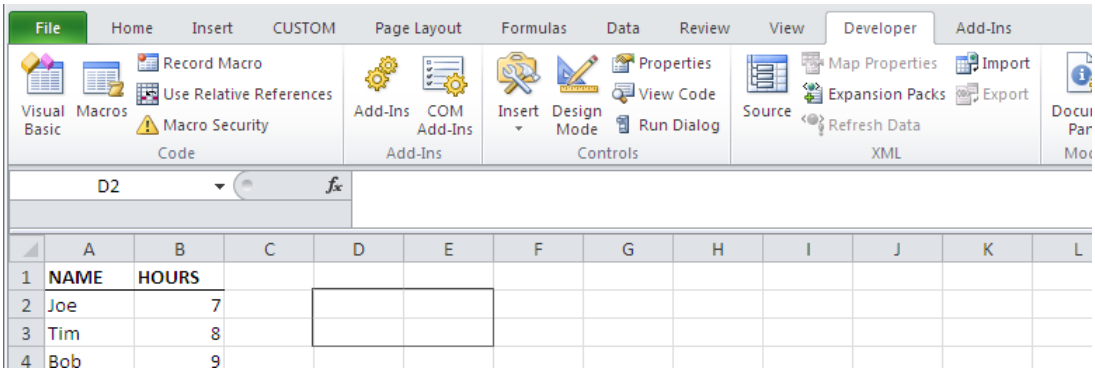
Assigning a macro to a BUTTON

Once you are satisfied with the macro and it's perfomance, you may assign that macro to a button within the spreadsheet.

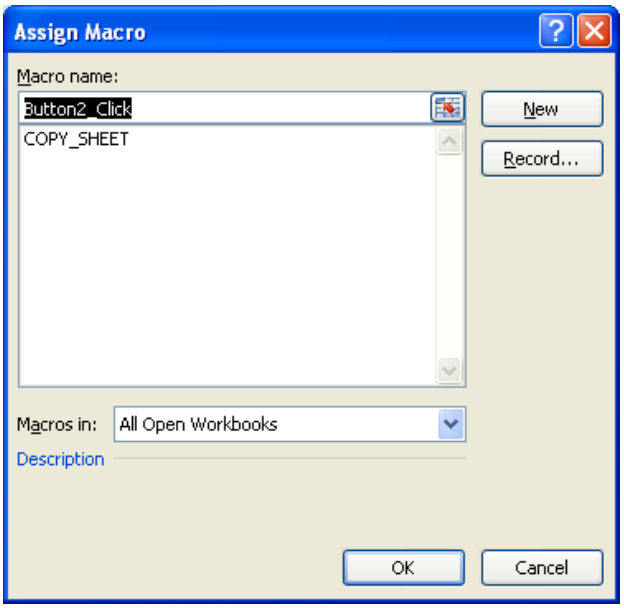
Access the Developer tab – select INSERT – Form Controls – Button (Form Control)



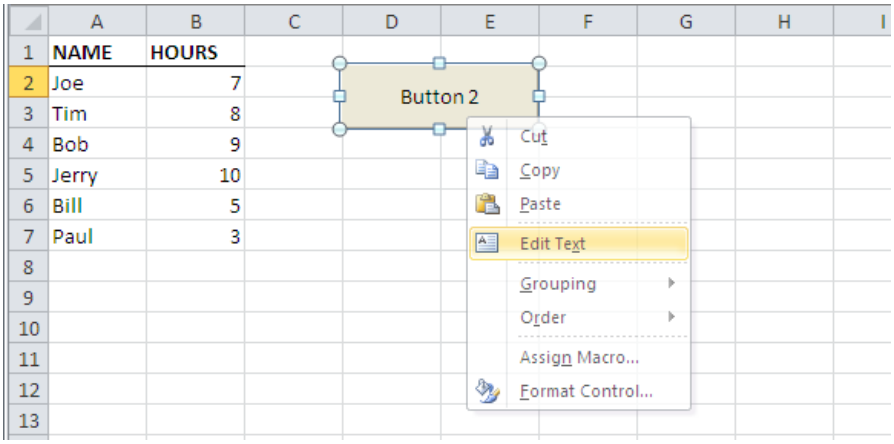
This will allow us to create a button and assign the macro to the button. You will first be prompted to "draw" the size of the button. Click and drag the cursor to create the button shape and size. Release the mouse button.



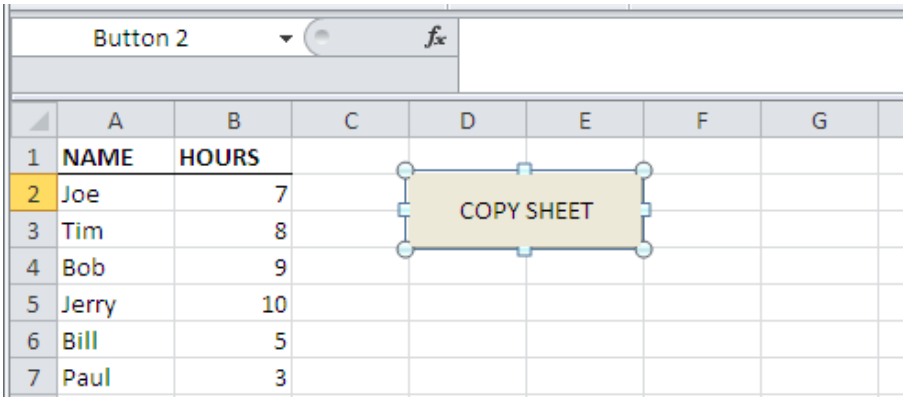
Select an existing macro – or yo may record a new macro from this step. We will choose the COPY_SHEET macro that we created earlier.



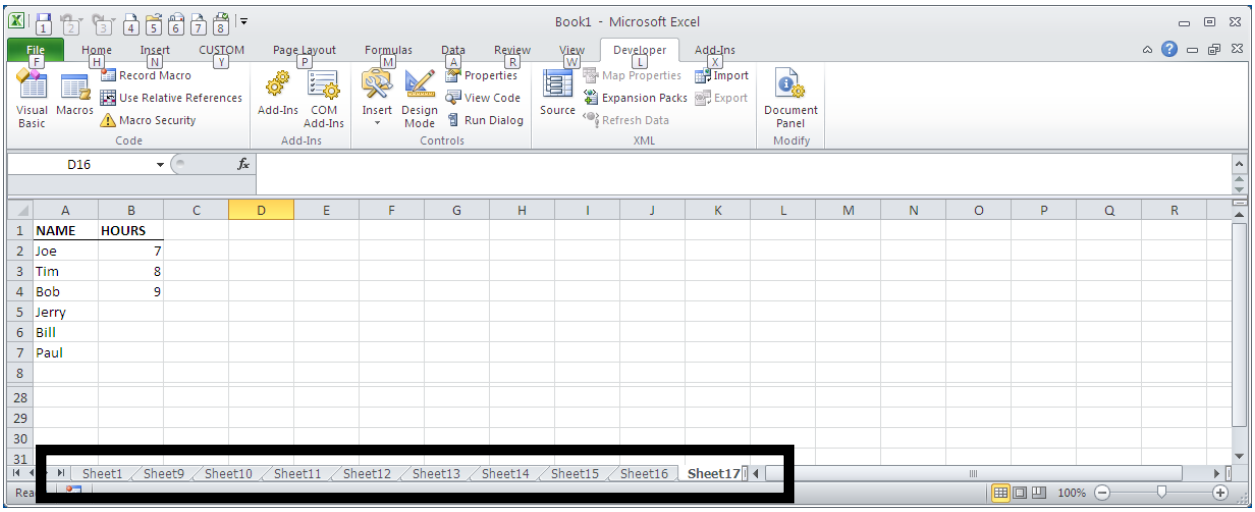
Right Click on the new button for additional options.



Select Edit Text to Change the text on the button.



Once this is complete, click the button to test the assignment of the macro.



Editing Existing Code (continued)

There is a nice bit of code that is used to eliminate the screen flicker while the macro runs. This code disables the screen updating. Here is an example of the code.

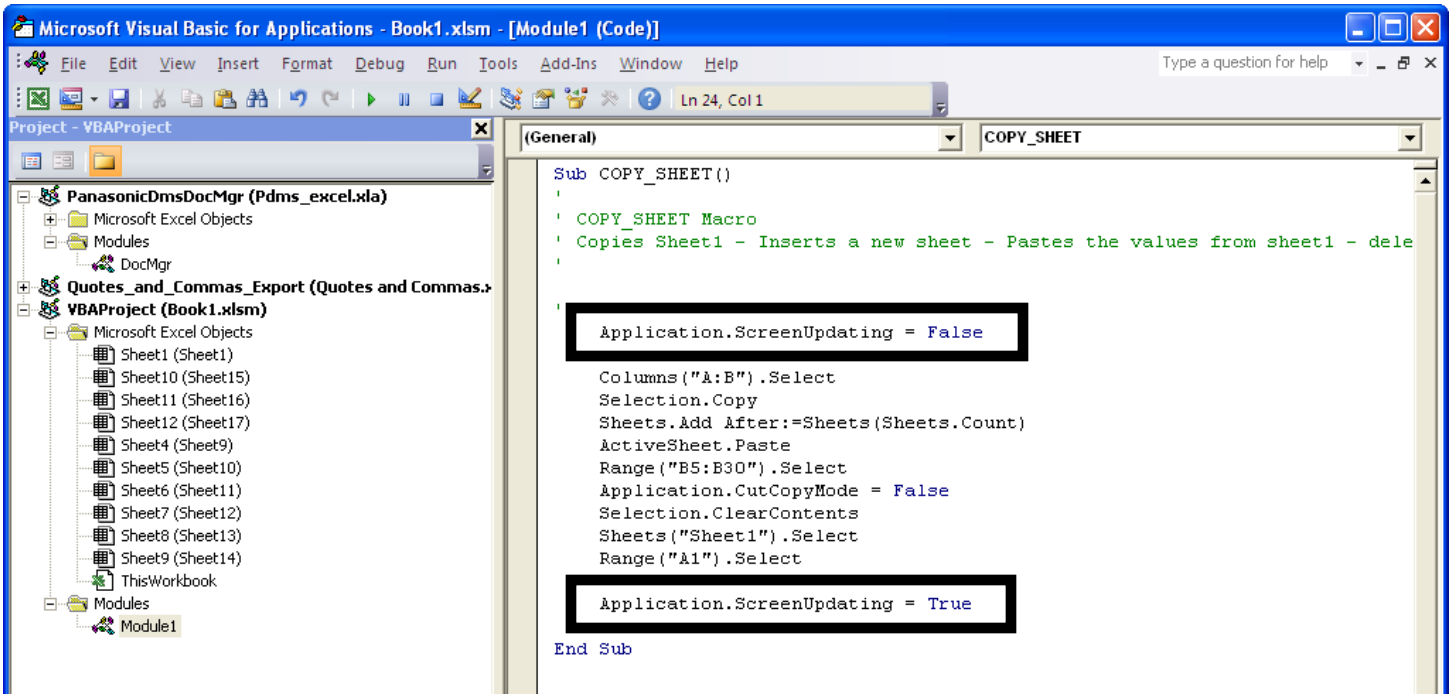
```
Sub Macro1()

'

        Application.ScreenUpdating = False

        'YOUR CODE

        Application.ScreenUpdating = True

End Sub
```

Excel **will** set screen updating back to **True** whenever focus is passed back to Excel (your macro finishes) in most cases, it pays to play it safe and include the code at the end.

Let us edit the Copy_Sheet macro and insert this text within the code.

From the Developer Tab, Choose MACRO – select the appropriate macro – click EDIT.

Type the code into the existing macro – or Copy and Pate the text into the code.



Close the VB editor and test the macro. You will notice that the code now runs without skipping through the process of executing the commands visually. This is a great way to speed up the execution of macros.

Here is an exampe of a Macro that includes :

- Protecting / Unprotecting Sheets (with password)
- Hiding / Unhiding Sheets
- Refreshing Queries
- Refreshing Pivot Tables
- Filtering out 0.00 values from a table

```
Sub Macro1()
'
' Macro1 Macro
' Stops screen updating while macro runs (will not show sheet unhide / hide)
    Application.ScreenUpdating = False
 ' Unprotects REPORT Sheet
   Sheets("REPORT").Unprotect Password:="XXXYYY"
' Unhides worksheet to refresh
   Sheets("JOBS").Visible = True
   Sheets("JC DATA").Visible = True
   Sheets("BUDGET").Visible = True
   Sheets("CO").Visible = True
   Sheets("PIVOT").Visible = True
' Refreshes Queries
   Sheets("CO").Select
   Range("A1").Select
   Selection.ListObject.QueryTable.Refresh BackgroundQuery:=False
   Sheets("BUDGET").Select
   Range("A1").Select
   Selection.ListObject.QueryTable.Refresh BackgroundQuery:=False
   Sheets("JC DATA").Select
   Range("A1").Select
   Selection.ListObject.QueryTable.Refresh BackgroundQuery:=False
   Sheets("PIVOT").Select
```

```
    Range("A4").Select
    ActiveSheet.PivotTables("PivotTable1").PivotCache.Refresh
' Hides worksheets
    Sheets("JOBS").Visible = False
    Sheets("JC DATA").Visible = False
    Sheets("BUDGET").Visible = False
    Sheets("CO").Visible = False
    Sheets("PIVOT").Visible = False
' Unhides all Rows in MasterTable
    Sheets("REPORT").Select
    ActiveSheet.ListObjects("Table2").Range.AutoFilter Field:=8
    Range("A5").Select
' Filters out all rows with 0.00 value in TOTAL column
    ActiveSheet.Range("$A$7:$M$705").AutoFilter Field:=8, Criteria1:="<>0", Operator:=xlFilterValues
' Protects REPORT sheet with password
    Sheets("REPORT").Protect Password:="XXXYYY"
End Sub
```

NOTES ON MACROS :

- Something as innocent as renaming a sheet can cause a macro to fail. When created, a macro is very specific to select certain worksheets within a workbook. If you change the name of a worksheet on a particular workbook, the code will lose reference to the original name, and break.
- Google is your friend. You can use Google searches to find almost anything. Use a search string that includes “MACRO” at the end of whatever you are looking for. For example: “Hide specific worksheets macro”, “Rename sheets based on cell value MACRO” or “find last row MACRO”. The search results will take you to a number of websites and message boards with people who have the same problem(s). Of course, use caution when using code written by somebody else. Review the code and “make it yours”.
- Get to know the basics of VB Code – you don’t necessarily have to know how to WRITE the code, but understanding the code will help with creating extremely effective macros.
- Keep in mind that the ORDER in which items are recorded may cause problems. If a macro is not working, try switching the order in which the macro is recorded.
- For larger projects, try to record smaller portions of the macro. Run the small parts and paste the macro together in pieces, testing the results as you go along.

```
Sub Macro3()
'
' Macro3 Macro
'
    Application.ScreenUpdating = False
' Refreshes Queries
    Sheets("JOB").Select
    Range("A1").Select
    Selection.ListObject.QueryTable.Refresh BackgroundQuery:=False
    Sheets("EMPLOYEES").Select
    Range("A1").Select
    Selection.ListObject.QueryTable.Refresh BackgroundQuery:=False
    Sheets("TCHIS").Select
    Range("A1").Select
    Selection.ListObject.QueryTable.Refresh BackgroundQuery:=False
    Sheets("REPORT").Select
    Range("A3").Select

' Refreshes Pivot Table
    ActiveSheet.PivotTables("PivotTable1").PivotCache.Refresh
    Range("A1").Select

'  Creates a new Tab for Every Date in Pivot Table
    ActiveSheet.PivotTables("PivotTable1").ShowPages PageField:="CHECK DATE"
    Range("D1").Select
```

```
' Renames New Sheets as Date in Cell B1
    Dim sh

  For Each sh In ActiveWorkbook.Worksheets

   If IsDate(sh.Range("B1")) Then
     sh.Name = Format(sh.Range("B1").Value, "MM-DD-YY")
   Else
     sh.Name = sh.Name
   End If
  Next sh

' Autosize all Worksheets Workbook
  Dim wkSt As String
  Dim wkBk As Worksheet
  wkSt = ActiveSheet.Name
  For Each wkBk In ActiveWorkbook.Worksheets
    On Error Resume Next
    wkBk.Activate
    Cells.EntireColumn.AutoFit
  Next wkBk
  Sheets(wkSt).Select
  Application.ScreenUpdating = True


End Sub
```

```
Sub Delete_Sheets()
'


  Dim s


  Application.DisplayAlerts = False
  For Each s In Sheets
    If s.Name <> "Date" And s.Name <> "REPORT" And _
      s.Name <> "TCHIS" And s.Name <> "EMPLOYEES" And _
      s.Name <> "JOB" Then
        s.Delete
    End If
  Next
  Application.DisplayAlerts = True


'
End Sub
```