Learning Excel VBA

The Expense Report Macro Project

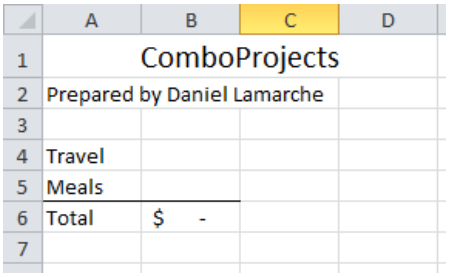
Prepared By Daniel Lamarche
ComboProjects

ComboProjects
Understanding does matter

The ‘Expense Report’ macro project

By Daniel Lamarche (Last update July 2019).

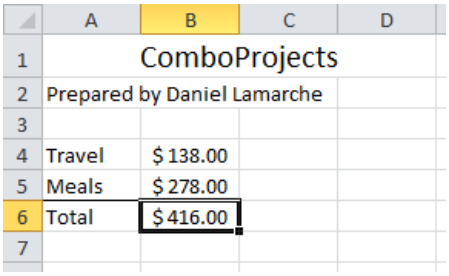
In this project we will create a macro that prepares a short expense report. We will see the limitations of the Macro Recorder, what it can do and what it cannot do. We will mostly focus on optimizing the code generated by the macro recorder.



	A	B	C	D
1	ComboProjects			
2	Prepared by Daniel Lamarche			
3				
4	Travel			
5	Meals			
6	Total	$ -		
7				

On the right you see an important stage of this macro: the steps that can be recorded. The features that cannot be implemented using the Macro Recorder will be added manually.

Once completed, the user needs only to enter the figures for the Travel and Meals expenses as shown here.



	A	B	C	D
1	ComboProjects			
2	Prepared by Daniel Lamarche			
3				
4	Travel	$ 138.00		
5	Meals	$ 278.00		
6	Total	$ 416.00		
7				

Following are the formatting features used in this project:

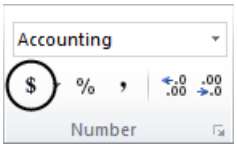
- Company name uses Merge and Centre.
- Automatically adds the author of the report and renames the worksheet using that name.
- Cells B4:B6 are formatted using the Accounting Number Format.
- A Bottom Border is applied to cells A5:B5.

Rehearse the macro

Depending on the complexity of your macro, it is strongly recommended that you rehearse all the steps before you record them using the **Macro Recorder**. This way you’ll avoid any hesitations on the day of you ‘live performance’.

In a new worksheet follow the steps below:

- Click cell A1 and type the name of your company. Then hit Ctrl+Enter (to stay in A1).
- **Merge and Centre** cells A1:D1.
- Increase the font size of the merged cells to 16 pts.
- In B2 type ‘Prepared by <then type your name>.
- In A4, A5 and A6 type ‘Travel’, ‘Meals’, then ‘Total’.
- In B6 click **AutoSum** and select B4:B5. The formula is =SUM(B4:B5).
- Format B4:B6 using the **Accounting Number Format**.
- Format cells A5:B5 using the **Bottom Border**.
- Click in cell B4. This is where the user starts entering the amounts.



Now that we are familiar with each steps it’s time to record them with the Macro recorder.

Let’s ensure that you have the Macro Recorder handy. Right-click the Excel’s Status Bar (at the very bottom the screen), ensure that **Macro Recording** is checked otherwise click the option as shown below.

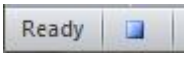
The Macro Recording icon now appears in the Status Bar immediately next to Ready.

Record your report

To start recording your expenses report, follow these steps:

- Click **Macro Recording**.
- In the **Macro name** box type 'MyReport'.
- In the **Shortcut key** box type an uppercase 'M'.
- In the **Store macro in** list leave it to 'This Workbook'.
- In the **Description** type an optional small comment.
- Click **OK**.

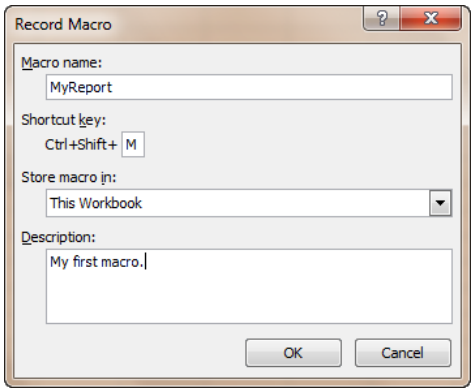
The **Macro Recording** turns into a blue square. Click this button when you are done recording all the steps for your macro.



Carefully execute all the steps we rehearsed earlier. Once you're completely done click the **Stop Recording** button.

The Record Macro dialog box

There are a couple of things you should know about the Record Macro dialog box.

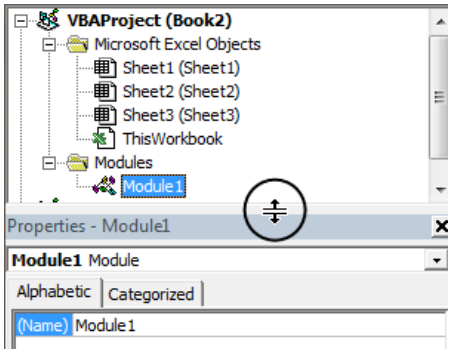


- The name of the macro cannot contain any punctuation characters and that includes spaces. You may use the underscore. For example you could have used 'My_Report'.
  Also a macro name cannot start with a number.
- If you assign a keyboard shortcut to your macro you should be aware that this shortcut will override any shortcut in Excel. So Ctrl+C would not be a good choice! Use that feature sparingly.
- A macro stored in 'This Workbook' will only be accessible in the current workbook. If you want your macro to be universally accessible store it in the 'Personal Macro Workbook'.

Viewing the macro

To view your macro hit Alt+F11. This will show the Visual Basic Editor (VBE).

On the left side of the VBE we need to open the **Projects** Pane so hit Ctrl+R to view that panel. We will also need to see the **Properties** pane. Hit the F4 key to view that panel.



You will probably need to resize the **Properties** pane a bit so you can see more of the **Project** pane. As shown on the right, reduce the height of the **Properties** panel about half the height of your screen.

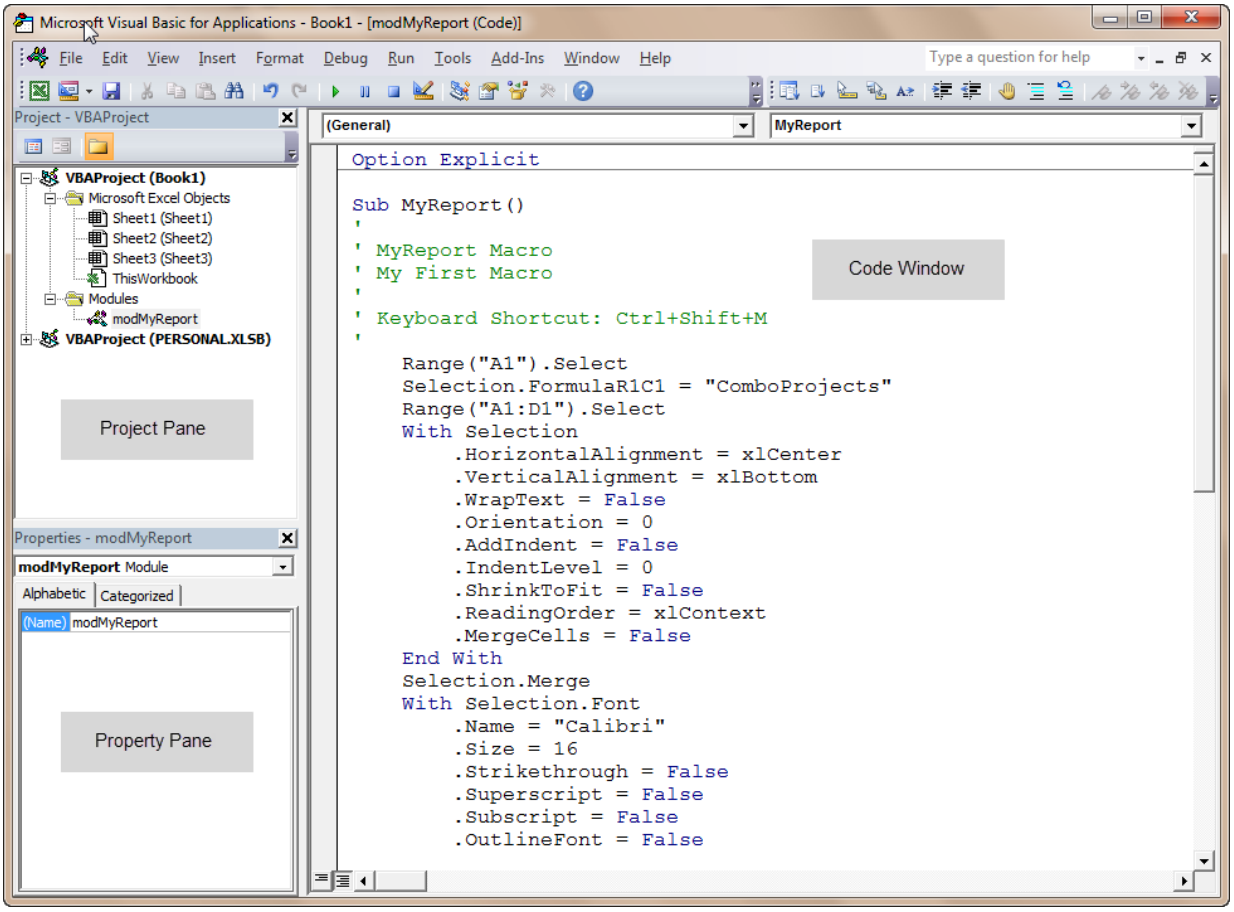
In the **Project** panel expand the **Modules** folder so that you can see your module(s). Double-click the module **Module1** to open it in the **Code** Window.

Renaming your module

For the same reason you rename worksheets in Excel, you should rename all your modules with a descriptive name. To rename a module you 'unfortunately' need to use the **Properties** panel.

In the **Name** properties replace Module1 with the 'modMyReport' and hit Enter. As with macro names you cannot use any punctuation. Also, the first character cannot be a number.

Your VBE screen should look like the screenshot below.



If you scroll down in the Code Window you should see tons of lines of code. If you think that there is quite a lot of statements for such a small report then you're perfectly right. We'll need to heavily amend the code.

Not only the Macro Recorder 'records' all your steps but it add many useless lines that you do not need. Also the *mechanics* of the recorded macro is not optimized at all.

Optimizing a recorded macro is done in three phases described below:

- Remove all default properties.
- Remove all **Select – Selection** pairs. They may also have the form **ActiveCell – Selection**.
- If you have any **With – End With** pairs they probably need adjustments.

Understanding properties

In general a property has the form **Objet.Property = Value** where Object is an Excel component and Property is a formatting feature or an attribute for that component. Below are two simple examples:

```
Range("A1").Font.Size=16
ActiveCell.FormulaR1C1 = "=R[-2]C * 10"
```

In the first example Range("A1") is the object and .Font.Size are properties. In the second example ActiveCell is the object and FormulaR1C1 is the property. One way to quickly recognize a property is that the property will often be made equal to a value. One last example:

```
Sheets(3).Name="March"
```

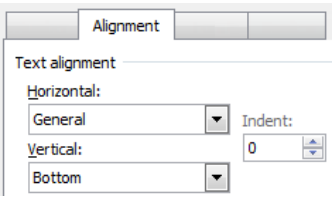
Here Sheets(3) is the object (the third worksheets in the workbook) and Name is the property and it is made equal to the text 'March'.

Phase I — Removing Default Properties

Default properties are, for example, formatting features that are in your code but you haven't use while recording your macro. They represent a default value because a cell or selection is formatted that way in Excel by default. For example look at a sample code taken from the beginning of the macro:

```
With Selection
.HorizontalAlignment = xlCenter
.VerticalAlignment = xlBottom
.WrapText = False
.Orientation = 0
.AddIndent = False
.IndentLevel = 0
.ShrinkToFit = False
.ReadingOrder = xlContext
.MergeCells = False
End With
```

By default a cell's Horizontal Alignment is **General**. Here the property HorizontalAlignment equal **xlCenter**. Since Centre is not a default setting for a cell you need to keep that line of code. Actually this Centre value is there because we used Merge and *Center* in our report.



You can verify this by using the Format Cells command on a cell that has never been used. In the Alignment tab you will see that Horizontal format is set to General.

All other properties in the sample code above were never used while recording the macro so they can go. A quick way to remove a line in the code window is to move the insertion point anywhere in a line and press Ctrl+Y. Remove all the lines except the for .HorizontalAlignment property.

Let's look at another group of lines:

```
With Selection.Font
.Name = "Calibri"
.Size = 16
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
.Shadow = False
.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor
```

```
End With
```

Looking at these properties you notice that the only property that you used while recording the macro what the font size for the company name. All other properties either use the default Excel format (like the font name) or they are set to False, zero or some form of the expression *not used*.

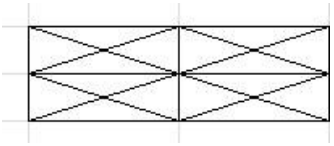
Remove all lines except for the second line specifying the Font Size to 16.

Further down there is a section with 8 lines starting with Selection.Borders. As you can clearly see 7 of them are set to xlNone, meaning that they are not used.

About borders

In our macro, we formatted 2 cells using Bottom Borders. Since there are 8 different borders setting available in Excel but we used only the bottom border the macro recorder show all eight but only add some useful code to the xlEdgeBottom property.

If you're curious about the 8 types of borders available look at the image on the right. See if you can identify each of them. If you need a hint examine your code for the various parameters of the eight Border properties.



Delete all seven **Borders** properties that are set to xlNone.

Phase 2 — Remove all 'Selection – Select' pairs

About half of the code is now gone. This makes the code easier to read and run faster. Let's further optimize the code. Look that the beginning of the macro where you type the company name. It is made of two statements.

```
Range("A1").Select
Selection.FormulaR1C1 = "ComboProjects"
```

The first statement specifies that the macro must move to cell A1 then type the company name. You may think that this is fine but it is not!

In real life a user needs to go to a cell before he/she types an entry. However, in VBA you do not need to go to a cell in order to add a value in it. Think about it, the second statement uses the object Selection. If you were asked what cell is currently selected you will immediately reply "Of course it's A1".

So instead of going to cell A1 then type some text we're going to instruct VBA to type the entry *in cell* A1, without moving the active cell.

Start deleting before the .Select until the line below moves up and continue until you get to the equal character then type .Value (note the full stop in front of Value?). The final result is:

```
Range("A1").Value = "ComboProjects"
```

This way VBA says to Excel to type an entry is that cell instead of 1) move to the cell and 2) type in it. This takes much less time to execute. Modify the two lines where you typed 'Prepared by …' to read

```
Range("A2").Value = "Prepared by type your name"
```

What's the rule?

When a line ending with .Select and is followed by **ActiveCell** or **Selection** then, clearly these two properties refer to the location in front of the .Select. Delete the **Select** and its corresponding

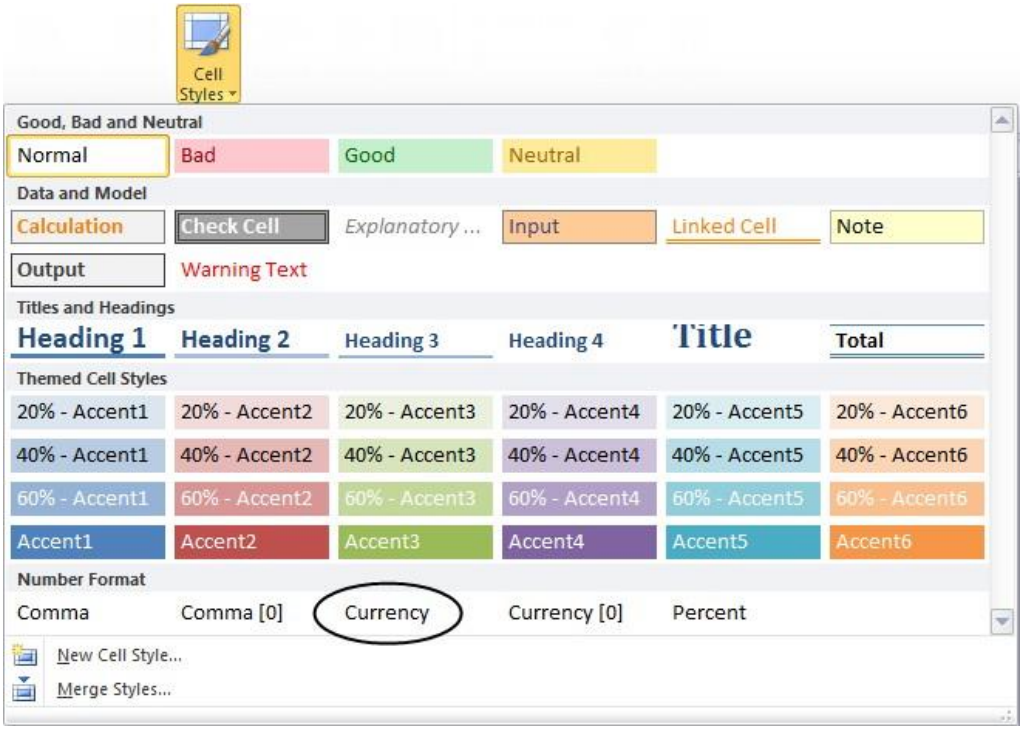
ActiveCell or **Selection**. Below is the final result for the next 5 lines of code. Please modify them as shown:

```
Range("A4").Value = "Travel"
Range("A5").Value = "Meals"
Range("A6").Value = "Total"
Range("B6").FormulaR1C1 = "=SUM(R[-2]C:R[-1]C)"
Range("B4:B6").Style = "Currency"
```

This technique allows you to work in any cell remotely. Now it takes a fraction of the time for the macro to enter all these values.

The macro recorder will use the property .FormulaR1C1 for all entries. You can safely replace it with .Value instead. On the other hand you see that the second last line is indeed a formula using R1C1 notation so leave it as shown. Finally the last statement uses the **Currency** style for the cells B4:B6 so keep the .Style property.

Below you see the **Cell Style** command off the **Home** tab with the **Currency** style highlighted at the bottom. This is where VBA takes its information regarding the Style property used in the code.



Whenever you see a Range or a Selection object using the .FormulaR1C1 property for any types of entry (numeric or text) other than a formula using the R1C1 notation you can safely replace it with Value.

Update the line using your company name with .Value instead of .FormulaR1C1.

Phase 3 — The Finale

Near the top of the code you will find the following lines:

```
Range("A1:D1").Select
With Selection
    .HorizontalAlignment = xlCenter
End With
```

```
Selection.Merge
With Selection.Font
    .Size = 16
End With
```

Here we see a new VBA structure: the **With – End With** pair. We also see a couple of occurrence of the **Selection** object. And we want to eliminate all useless references to **Selection** right?

If you examine carefully that bit of code you'll agree that all **Selection** objects refer to Range("A1:D1"). We will address this in a minute. For the moment let's look at the meaning of **With – End With**.

The With – End With construct

Whenever you see a **With – End With** statements this is a good news. It is a powerful way to optimize your code. Using a **With – End With** construct you can change as many properties for an object as required without re-qualifying the object for every property. Let's explain.

Look at the following dummy example:

```
Range("A6").Value = "Total"
Range("A6").HorizontalAlignment = xlCentre
Range("A6").Borders(xlEdgeBottom).LineStyle = xlContinuous
Range("A6").ThemeColor = xlThemeColorLight2
Range("A6").Font.Bold = True
```

Instead of qualifying Range("A6") for each statement, it would make much more sense to do it like this:

```
With Range("A6")
    .Value = "Total"
    .HorizontalAlignment = xlCentre
    .Borders(xlEdgeBottom).LineStyle = xlContinuous
    .ThemeColor = xlThemeColorLight2
    .Font.Bold = True
End With
```

Can you see how this would run much faster than the previous example? Let alone much less typing! In the first example, VBA had to resolve each instance of Range("A6"). In the later example this is done once. As you can guess, there is no limit to the number of properties you can update for an object when you use the **With – End With** structure.

If necessary, you can even have a **With – End With** nested with another **With – End With**. For example if you had to change a couple of properties of a worksheet as well as a cell in that worksheet you could consider doing like that:

```
With Sheets(1)
    .Select
    .Name = "Summary"
    With .Range("A1")
        .Value = "This is a long entry."
        .Borders(xlEdgeBottom).LineStyle = xlContinuous
        .Columns.Autofit
    End With
End With
```

In a nested **With – End With** the object referred to in the With belongs to the parent object. In the example above the object **Range("A1")** refers to cells A1 in **Sheets(1)** of the current workbook.

The three properties **Value**, **Border** and **Columns** are automatically linked to cell A1 (since they start with the full stop) of the sheet referred to in the outer **With – End With**. Modify the top section of the MyReport macro to look as shown below. A new feature is added for the **Font** property.

```
With Range("A1:D1")
    .HorizontalAlignment = xlCenter
    .Merge
    .Value = "ComboProjects"
    With .Font
        .Size = 16
        .Italic = True    ' The Italic was added after the macro was recorded.
    End With
End With
```

Near the end of the procedure update the section:

```
Range("A5:B5").Select
With Selection.Borders(xlEdgeBottom)
    .LineStyle = xlContinuous
    .ColorIndex = 0
    .TintAndShade = 0
    .Weight = xlThin
End With
```

With the following:

```
With Range("A5:B5").Borders(xlEdgeBottom)
    .LineStyle = xlContinuous
End With
```

The last three properties in the original code above use their default values; there is no need to keep them. We'll only keep **LineStyle** because the default for a cell is to have its border set to xlNone.

Phase 4 — Adding Custom Features

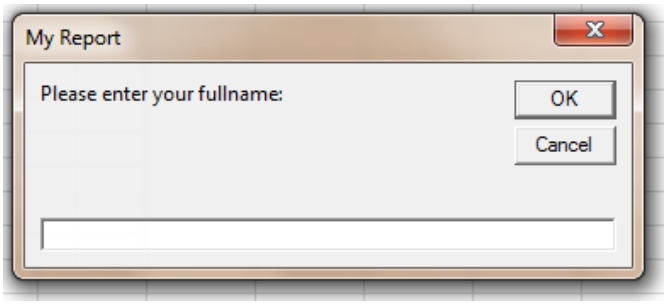
The customer calls you and wants to amend your macro by adding two features.

Since he will need to share this macro with 15 employees, the customer requests that instead of having 'Prepared by Daniel Lamarche' the macro should ask for the user name when it is launched. The name typed in the pop-up box is then used where we display 'Prepared by […]'. This way he only needs one macro for the 15 employees.

Below the name of the report's author the customer also requests to have the date the report is printed.

There is no way the macro recorder can accomplish these two tasks. It needs to be implemented manually.

Amend the code just below the comments at the top as shown:



```
Dim strEmployee As String      ' Create a text variable.
strEmployee = InputBox("Please enter your full name: ", "My Report")
```

A variable is created to store the name of the employee.

The **InputBox** function allows the staff to type their name. The name is then stored in the variable **strEmployee**.

You can see the result on the InputBox statement on the previous page.

Now amend the line:

```
Range("A2").Value = "Prepared by Daniel Lamarche"
```

To read:

```
Range("A2").Value = "Prepared by " & strEmployee
```

Finally the name captured in the InputBox is concatenated after the 'Prepared by ' string. Try the macro before we move to the last part.

To conclude we need to add the date the report is printed. The customer would like the see under the employee name something like:

Printed on 16-Jul-2016

The problem is that he also requires that there is a blank line below the date and the report. You sigh as you realize that you will need to modify the code where Meals, Travel, etc. so that it appears one row further down in Excel.

Modify your code so that all row numbers are increased by 1. For example A4 becomes A5. Do not miss any reference! Below is the final result.

```
Range("A5").Value = "Travel"
Range("A6").Value = "Meals"
Range("A7").Value = "Total"
Range("B7").Formula = "=SUM(B4:B5)"
Range("B5:B7").Style = "Currency"
With Range("A6:B6").Borders(xlEdgeBottom)
    .LineStyle = xlContinuous
End With
Range("B5").Select
```

Adding the printed date

Immediately after the line:

```
Range("A2").Value = "Prepared by " & strEmployee
```

Type the following line:

```
Range("A3").Value = "Printed on " & Format(Date, "dd-mmm-yyyy")
```

The **Format** function uses the current date and formats the date using 2 digits day, the abbreviated month name and a four digits year.

The Final Touch

Finally, since the figures may be in the thousands, we want to avoid having hash characters across the cell because the column is too narrow.

Add the final touch before the End Sub statement and run the macro.

```
Columns("B").ColumnWidth = 11
```

Running the macro in Break Mode

A cool way to see how each line of a macro runs in Excel is to split your screen in two — Excel on the left and VBE on the right.

Then click anywhere in your sub and press the F8 key. Every time you hit F8 while in your code VBA will execute one line at a time. This feature is called **Step-In**. You will be able to see how each line of code affects the worksheet.

Final Touch — Adding a Button to Run the Macro

To run the macro by clicking on a button you can use a shape. From the **Insert** tab find the **Shape** drop down menu and select the **Rectangle** shape.

Hold the Alt key while drawing the shape on the worksheet. The Alt key will cause the outline of the button to snap to the worksheet grid.

Once your rectangle in in place, right click on it and select **Assign Macro** to select the macro you want to run when the user click that shape. Select **MyReport** from the list. Finally, right-click the shape then type 'Expense Report'. Format the shape at will.

Note that if you click outside the button before you change the text in the button, clicking on it will run the macro! If all you want to do is rename the text in the button, right-click on the button and choose **Edit Text**.

Now click on the button and enter your name in the dialog box then click OK. Don't forget to shout WOW!

Last Hurray

Hopefully this tutorial have helped you to understand (or review) a couple of important concepts about planning a macro and, very important, optimizing it.

Daniel Lamarche
comboprojects.com.au