Excel

51 Awesome MACROS

Ready-to-use

Save time and be more productive

Save file in multiple directories

Find files in network with VBA

Send Email with Attachments

Login and password system

Find error's automatically

And much more...

Philippe A. Louis

Summary

HIDE AND SHOW INFORMATION

OTHERS

INTERACTION WITH WINDOWS

INTERACTION WITH OUTOLOOK

INTERACTION WITH POWERPOINT

INTERACTION WITH WORD

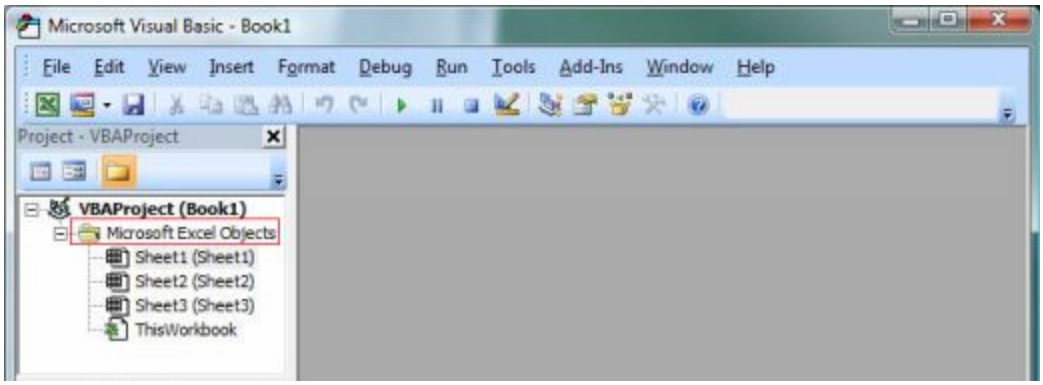
Introduction

This book approaches Excel with emphasize in VBA codes, there are 51 ready Macros to apply in thousands of scenarios. In order to get most out of this book, some basic and intermediate skills of Excel knowledge are necessary. This short introduction is for users who do not know the basic process of macro development, this is a brief of how insert the code in the VBA editor and how to run those routines.

There are three main steps to add macros in Excel:

1. Access the Developer superior tab , click the Visual Basic Button (Shortcut Alt + F11)

2. Select the object to allocate the macro, in most cases, the user can insert the code in a new module; by right click in the Microsoft Excel Objects folder, Insert, Module.



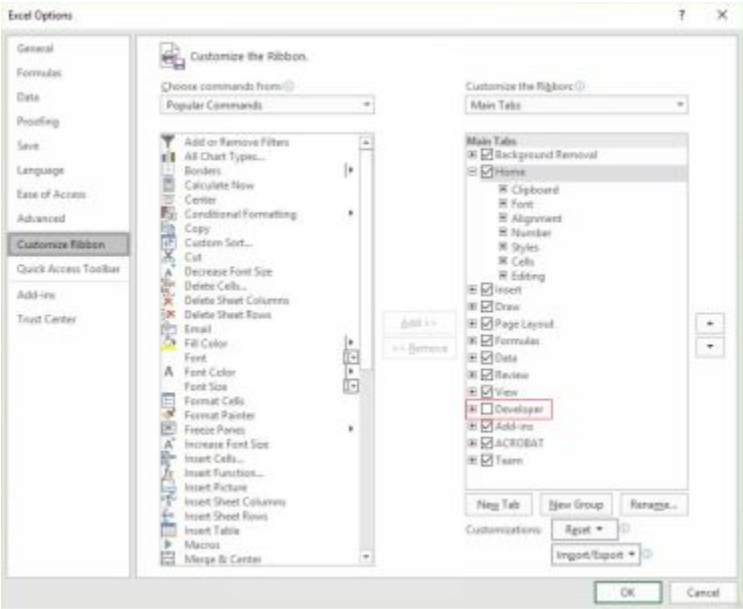
3. Insert the macro in the text box and run the code clicking in “Play” button, or pressing the shortcut F5. The user can also run the macro “Step by Step”, to verify each line of code, with the shortcut F8 to debug the code.

Obs 1. For some macros, it is necessary allocate the code in specific objects (Sheet's or ThisWorkbook object), for those cases, there are a note specifying how to proceed.

Obs 2. Once the macro is ready in VBA editor, just close the Visual Basic window. To access the macro again, can do it easily and quickly in the Developer superior tab and button Macros.

Obs 3: If the tab "Developer" is not available, just follow the steps bellow:

1) File
2) Options
3) Customize the Ribbon
4) In the selection, just check the option Developer



Open Save and Close

1. Open a spreadsheet, if it’s already opened, just maximize

This routine is useful when applied in process that frequently open’s external files. A simple code of “open file” will fail if the file was already open, what is a common problem. This routine prevent this error, by verifying if the file required is minimized, and then just maximize the window, else if, the macro will open and then maximize.

Obs. Change the example values highlighted in bold

```
Sub Openandmax()
Dim FileName, FolderPath, Extension As String
FolderPath = "C:\Users\usertest\Documents"
FileName = "Workbook1"
Extension = "xlsx"
Dim wkb As Workbook
On Error Resume Next
Set wkb = application.Workbooks(FileName)
If Err <> 0 Then
Set wkb = application.Workbooks.Open(FolderPath + "\" + FileName + "."
& Extension, UpdateLinks:=0)
End If
On Error GoTo 0
Workbooks(FileName).Activate
End Sub
```

2. Auto save before close

This macro is extremely simple but useful, with this routine; Excel always will save before close. This routine is good to avoid the alert message from Excel and also works to prevent accident of close without save.

Obs. Add this code to the "ThisWorkbook" code object module

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
Application.DisplayAlerts = False
ActiveWorkbook.Save
Application.DisplayAlerts = True
End Sub
```

3. Copy a worksheet, copy/paste values and save

It is one of my most used routines. It’s a really simple code, which is good for user’s who work with a lot of functions, but need’s to create a version just with values before release the report. This code will copy a worksheet, select all cells, copy and then paste values to eliminate all sorts of formulas. This macro also will save the file in a chosen path.

Obs. Change the example values of color highlighted in bold

```
Sub CopiarSaida()
Dim iPath, iFileName, iSheet As String
iSheet = "Plan3"
iPath = "C:\Users\UserTest\Documents\"
iFileName = "Planilha.xls"
Worksheets("Sheet1").Copy
Cells.Copy
Cells.PasteSpecial Paste:=xlPasteValues
ActiveWorkbook.SaveAs Filename:= iPath & iFileName, _
FileFormat:=xlOpenXMLWorkbookMacroEnabled
End Sub
```

4. Save fast backup

Backups are important for files safety; this routine was created to allow users create a fast backup in a new file, with the current path and name concatenated with date and time.

```
Sub SaveBackup()
```

```
ThisWorkbook.SaveCopyAs Filename:=ThisWorkbook.Path & _
"\" & Format(Date, "mm-dd-yy") & " " & _
ThisWorkbook.Name
End Sub
```

5. Auto save a backup copy before close

This macro automatically saves a backup copy of the file before close. The backup file receive the current time and data in file name.

Obs. Add this code to the "ThisWorkbook" code object module

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
Application.DisplayAlerts = False
FileDefaultName = "TestFile"
Application.DisplayAlerts = False
ActiveWorkbook.SaveAs Filename:=Application.ActiveWorkbook.Path &
"\" & Format(Time, "hhmmss") & " " & Format(Date, "mm-dd-yy") & " "
& FileDefaultName
Application.DisplayAlerts = True
End Sub
```

6. Save each sheet as a new file

This code is simple, and it’s works to save each worksheet in different file in a chosen path.

Obs. Change the example values highlighted in Bold.

```
Sub SaveSheets()
Dim wkb, iFileNames As String
wkb = ActiveWorkbook.Name
PathFolder = "C:\DestinyFolder"
For i = 1 To Worksheets.Count
Worksheets(i).Select
iFileNames = ActiveSheet.Name
```

```
ActiveSheet.Copy
ActiveWorkbook.SaveAs Filename:= PathFolder + "\" + iFileNames + "."
& "xlsx"
ActiveWorkbook.Close
Workbooks(wkb).Activate
Next
MsgBox "File Saved with Success"
End Sub
```

7. Save each sheet as PDF file

This code was made for save each worksheet as a PDF file in a chosen path.

```
Sub SaveWorkshetAsPDF()
Dim ws As Worksheet
PathFolder = "C:\DestinyFolder"
For Each ws In Worksheets
On Error Resume Next
ws.ExportAsFixedFormat xlTypePDF, PathFolder & "\" & ws.Name &
".pdf"
Next ws
End Sub
```

8. Save the file in different directory's

It is usual to have necessities of save the same file in different paths. Through this macro, it will be possible in an easy and fast way.

Obs 1 Change the name of file, extension and folder path, highlighted in bold.

Obs .: For change, erase or include new path's, just need to modify the example. For inclusion, just add FolderPath(4), FolderPath(5) and so on.

```
Sub SaveFile()
Dim NomeArquivo, Extensao As String
```

```
Dim iFolder(1 To 999) As String
iFileName = "File Name"
iExtension = "xls"
iFolder(1) = "C:\DestinyFolder"
iFolder(2) = "C:\DestinyFolder"
iFolder(3) = "C:\DestinyFolder"
For i = 1 To 999
   If iFolder(i) = "" Then
      GoTo ExitShortcut:
   End If
   Perg = MsgBox("Save file as: " & iFileName & " in the folder: " &
iFolder(i), vbYesNo)
   If Perg = vbYes Then
      ActiveWorkbook.SaveAs Filename:=iFolder(i) + "\" + iFileName + "."
& iExtension
   End If
Next i
ExitShortcut:
MsgBox "File Saved with Success"
End Sub
```

9. When close the file, inform the time spent

Routine control is a common practice inside companies, this code helps users control their time, by notifying the user how many time were spent with the file, from the moment it’s was opened until close.

Obs. Add this code to the "ThisWorkbook" code object module

```
Option Explicit
Dim StartTime As Date
Dim EndTime As Date
Dim Elapsed As Double
Dim iMinutes As Long
Dim dblSeconds As Double
Private Sub WorkBook_Open()
```

```
StartTime = Now
End Sub
Private Sub WorkBook_BeforeClose(Cancel As Boolean)
EndTime = Now
Elapsed = 86400 * (EndTime - StartTime)
If Elapsed < 60 Then
   MsgBox "This file has been used for: " & Format(Elapsed, "#0.0") & "
Seconds", vbInformation + vbOKOnly
   Exit Sub
Else
   On Error GoTo ShortcutA:
   iMinutes = Elapsed / 60
   On Error GoTo ShortcutA:
   dblSeconds = Elapsed - (60 * iMinutes)
   MsgBox " This file has been used for: " & Format(iMinutes, "#") & ":" &
Format(dblSeconds, "00") & " Minutes", vbInformation + vbOKOnly
   Exit Sub
End If
ShortcutA:
MsgBox "When you open the file again, the time will be recorded"
End Sub
```

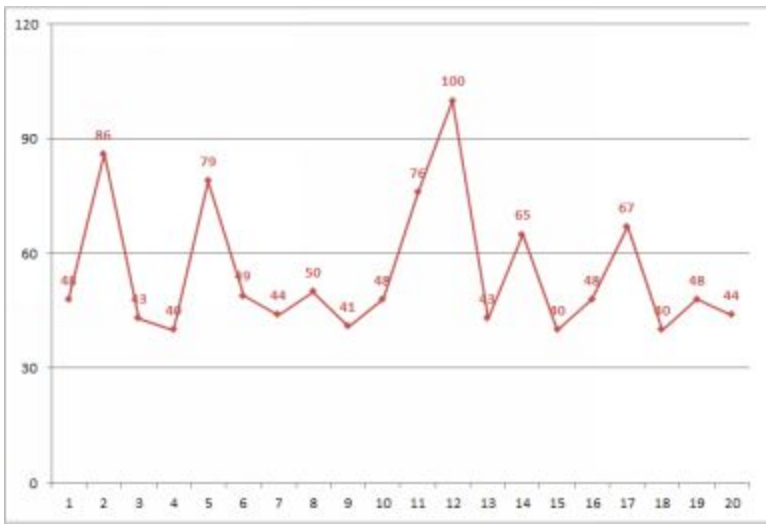
Interaction with Charts

0. Automatically adjust chart label's

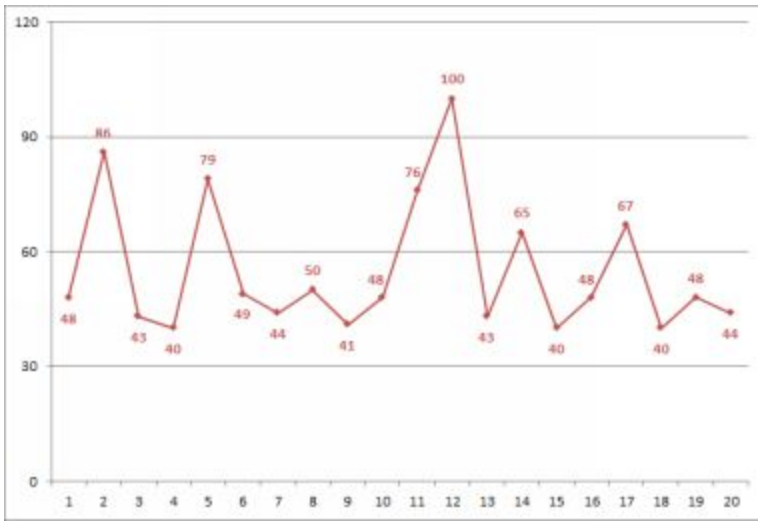
This code will automatically adjust labels considering the direction of the chart line, as the image bellow:

Obs. The code will adjust all charts in the active worksheet

Before:



After:



```
Sub AjustGraphic()
For i = 1 To ActiveSheet.ChartObjects.Count
ActiveSheet.ChartObjects(i).Select
Dim MaxScale, MinScale, MyPoint, DefaultPosition, AdjustedPosition As
Long
Dim MySeries As Series
Dim PointsArray As Variant
With ActiveChart
   MaxScale = .Axes(xlValue).MaximumScale
   MinScale = .Axes(xlValue).MinimumScale
   For Each MySeries In .SeriesCollection
      If MySeries.ChartType <> xlColumnClustered And _
         MySeries.ChartType <> xlLine And _
         MySeries.ChartType <> xlLineMarkers Then
         GoTo NEXTSERIES
      End If
         PointsArray = MySeries.Values
      For MyPoint = LBound(PointsArray) To UBound(PointsArray)
            If MySeries.Points(MyPoint).HasDataLabel = False Then
            GoTo NEXTDOT
            End If
            If MySeries.ChartType = xlColumnClustered Then
               MySeries.Points(MyPoint).DataLabel.Position =
xlLabelPositionOutsideEnd
               If PointsArray(MyPoint) > MaxScale * 0.9 Then
                  MySeries.Points(MyPoint).DataLabel.Position =
xlLabelPositionInsideEnd
               End If
            End If
            If MySeries.ChartType = xlLine Or MySeries.ChartType =
xlLineMarkers Then
               MySeries.Points(MyPoint).DataLabel.Position = xlBelow
               If MyPoint > 1 Then
               If PointsArray(MyPoint) > PointsArray(MyPoint - 1) Then
                  MySeries.Points(MyPoint).DataLabel.Position = xlAbove
               Else
```

```
                MySeries.Points(MyPoint).DataLabel.Position = xlBelow
            End If
        End If
        If PointsArray(MyPoint) > MaxScale * 0.9 Or _
            PointsArray(MyPoint) < MinScale * 1.5 Then
            MySeries.Points(MyPoint).DataLabel.Position = xlRight
        End If
    End If
NEXTDOT:
    Next MyPoint
NEXTSERIES:
  Next MySeries
End With
Next
End Sub
```

1. Resize all charts

Resize each charts in a worksheet can be an annoying task. This macro makes this process easy, resizing all charts with a chosen width and height.

Obs. Change the example values highlighted in bold

```
Sub RedmCharts ()
Dim i As Integer
For i = 1 To ActiveSheet.ChartObjects.Count
With ActiveSheet.ChartObjects(i)
.Width = 300
.Height = 200
End With
Next i
End Sub
```

Excel Links Management

2. Refresh all links

This macro was created for users who handle with large amount of information's. Depending how many functions are applied in a single worksheet's, links won't be automatically refreshed. With this routine, all links will refresh with a single click of mouse.

```
Sub LinksUpdate()
Dim linksarray As Variant
   linksarray =
ActiveWorkbook.LinkSources(Type:=xlLinkTypeExcelLinks)
   For i = 1 To UBound(linksarray)
   ActiveWorkbook.UpdateLink Name:=linksarray(i),
Type:=xlLinkTypeExcelLinks
   Next i
End Sub
```

3. Break all worksheets link

Sometimes when it's necessary send a report for other users it is recommended break all links with external sources in Excel. This is a simple macro that breaks all links.

```
Sub BreakLinks()
Dim alinksarray As Variant
   alinksarray =
ActiveWorkbook.LinkSources(Type:=xlLinkTypeExcelLinks)
   Do Until IsEmpty(alinksarray)
      ActiveWorkbook.BreakLink Name:=alinksarray(1),
Type:=xlLinkTypeExcelLinks
      alinksarray =
ActiveWorkbook.LinkSources(Type:=xlLinkTypeExcelLinks)
   Loop
End Sub
```

4. Change Links

As the first one, this routine was created to handle with links, but in this scenario, the user can change the links between different files easily and quickly.

Obs. Change the values of path, what is highlighted with bold. File1 is the current link and File2 is the new link wanted.

```
Sub ChangeLinks()
Dim File1, File2 As String
Dim wkb, wkb2 As String
wkb = ActiveWorkbook.Name
File1 = " C:\Users\usertest\Documents\ Planilha1.xls"
File2 = " C:\Users\ usertest \Documents\ Planilha2.xls"
Application.DisplayAlerts = False
Workbooks.Open Filename:=Arquivo2, UpdateLinks:=0
wkb2 = ActiveWorkbook.Name
Workbooks(wkb).Activate
ActiveWorkbook.ChangeLink Name:= File1, NewName:= File2,
Type:=xlLinkTypeExcelLinks
Workbooks(wkb2).Activate
ActiveWorkbook.Close
Application.DisplayAlerts = False
Workbooks(wkb).Activate
End Sub
```

Spreadsheet Control

5. Track users who opened the Workbook

This macro is important for users who need to register each access in the workbook. Through this code, the information's like user name, date and time will be recorded inside a worksheet.

Obs. It's necessary create a worksheet called "**Access Control**", where data will be recorded. The user can also change in the code, the name of the sheet where the data will be recorded.

```
Sub Auto_Open()
Dim iLine As Integer
Dim iDate, iTime As Date
iDate = Date
iTime = Time
iLine = Application.WorksheetFunction.CountA(Worksheets("Access
Control").Range("A1", "A1048576")) + 1
Worksheets("Access Control ").Range("A" & iLine) =
Application.UserName
Worksheets("Access Control ").Range("B" & iLine) = iDate
Worksheets("Access Control ").Range("C" & iLine) = iTime
Columns(1).AutoFit
Columns(2).AutoFit
Columns(3).AutoFit
End Sub
```

6. Highlight all edited cells

This macro it's make for workbook control, with this code, all cells what are edited by the user will be highlighted in a chosen color.

Obs 1. It is necessary include the code inside the desired object, in the example bellow, the worksheet will highlight the changes inside “Sheet1”.

```
Option Explicit
Private SelectA As Object
Private Sub Worksheet_Change(ByVal Target As Range)
   Dim cell As Variant
   For Each cell In Target
      If SelectA.Exists(cell.Address) Then
         If SelectA.Item(cell.Address) <> cell.FormulaR1C1 Then
            cell.Interior.ColorIndex = 35
         End If
      End If
   Next
End Sub
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   Dim cell As Variant
   Set SelectA = Nothing
   Set SelectA = CreateObject("Scripting.Dictionary")
   For Each cell In Target.Cells
      SelectA.Add cell.Address, cell.FormulaR1C1
   Next
End Sub
```

7. Protect all worksheets

The macro bellow automatically protect all worksheets from any data accidentally or deliberately changing.

Obs. Change the example values highlighted in bold

```
Sub ProtectSheets()
Dim wksht As Worksheet
For Each wksht  In ActiveWorkbook.Worksheets
wksht.Protect Password:="password"
Next wsheet
End Sub
```

8. Unprotect all worksheets

The macro bellow automatically unprotect all worksheets.

Obs. Change the example values highlighted in bold

```
Sub unprotectsheets()
Dim wsheet As Worksheet
For Each wsheet In ActiveWorkbook.Worksheets
wsheet.Unprotect Password:="password"
Next wsheet
End Sub
```

9. Protect a single worksheet with password

This code, protect the worksheet with an input box that requires a password. This routine works without Excel protection resource, with a different way.

Obs 1. The value of variable **SheetProtected** is the protected worksheet
Obs 2. Password is the sequence of characters needed to access the sheet
Obs 3. Anchor is the worksheet that will be selected in case of wrong password
Obs 4. Add this code to the "ThisWorkbook" code object module with “SheetActivate”

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
SheetProtected = "Sheet3"
iPassword = "123456"
Anchor = "Sheet1"
If ActiveSheet.Name = SheetProtected Then
iReturn:
inputPassword = InputBox("Password")
   If iPassword = inputPassword Then
      Exit Sub
   Else
```

```
        iAnswer = MsgBox("Wrong password, try again ?", vbYesNo)
        If iAnswer = 6 Then
            GoTo iReturn:
        Else
            On Error Resume Next
            Worksheets(Anchor).Select
        End If
    End If
End If
End Sub
```

20. Prevent user's save the file

This macro prevent the file save, it is useful for prevent accidentally or deliberately saves.

Obs. Add this code to the "ThisWorkbook" object module

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)
Msgbox "You cannot save this file"
Cancel = True
End Sub
```

21. Simple login system to access the spreadsheet

This is a simple system of login and password for get access in the spreadsheet.

Obs 1. It is necessary change the example values highlighted in bold.
Obs 2. The code need the Auto_Open() declare to work.
Obs 3. To change and create new user's and password, just modify the example values, for adding new user's, just follow the sequence, User(4), User(5) and so on. For register the corresponding password, the process is the same, Password(4), Password(5) and so on.

```
Sub Auto_Open()
```

```
Dim iUser(1 To 999) As String
Dim iPassWord(1 To 999) As String
Dim UserOK As Boolean
Dim PassOK As Boolean
'User Register
'-------------------
iUser(1) = "Daiana"
iUser(2) = "Clark"
iUser(3) = "Bruce"
'-------------------
iPassWord(1) = "1345"
iPassWord(2) = "1234"
iPassWord(3) = "5367"
'-------------------
UserOK = False
PassOK = False
Shortcut2:
InputUser = InputBox("Type your user name: ")
For i = 1 To 999
   If iUser(i) = InputUser And iUser(i) <> "" Then
   UserOK = True
   GoTo Shortcut1:
   End If
Next
Shortcut1:
If UserOK = True Then
   InputPassword = InputBox("Welcome: " & InputUser & ", type your
password:")
   If iPassWord(i) = InputPassword Then
   Exit Sub
Else
   question = MsgBox("Wrong password, do want to try again ?", vbYesNo)
   If question = 6 Then
      GoTo Shortcut2:
   Else
      MsgBox "This file will close"
```

```
        ActiveWorkbook.Close
    End If
End If
Else
    question = MsgBox("Wrong user name, do want to try again ?",
vbYesNo)
    If question = 6 Then
        GoTo Shortcut2:
    Else
        MsgBox "This file will close"
        ActiveWorkbook.Close
    End If
End If
End Sub
```

Error Management

22. Verify all worksheets to find errors

Verify error's is something important, the macro scan all filled cells in each worksheet's, looking for usual errors like #REF!, #N/D, #DIV/0 and so on. The error will be located and then communicated through a message box, showing the cell address.

```
Sub CheckError()
Dim ws As Worksheet
Dim ra As Range
For Each ws In Worksheets
   For Each ra In ws.UsedRange
      ra.Select
      On Error Resume Next
      If IsError(ra.Value) Then
         MsgBox "Aba: " & ra.Parent.Name & Chr(13) & "Célula: " &
ra.Address
      End If
   Next
Next
End Sub
```

23. Verify a selection to find errors

This routine is similar with the code before, but instead looking for errors in all worksheets, this one just scan in a selection.

```
Sub CheckErrorSelection()
Dim ra As Range
For Each ra In Selection
   ra.Select
   If IsError(ra.Value) Then
```

```
      MsgBox "Sheet: " & ra.Parent.Name & Chr(13) & "Cell: " &
ra.Address
   End If
Next
End Sub
```

24. Verify all worksheets and count errors

Verify error's is something important, the macro scan all filled cells in each worksheet's, looking for usual errors like #REF!, #N/D, #DIV/0 and so on. The error will be located and then will be counted and then communicated through a message box.

```
Sub CheckErrorAllSheets()
Dim ws As Worksheet
Dim ra As Range
Dim iCounter As Long
iCounter = 0
For Each ws In Worksheets
   For Each ra In ws.UsedRange
      If IsError(ra.Value) Then
         iCounter = iCounter + 1
      End If
   Next
Next
MsgBox (iCounter & " Errors Founded")
End Sub
```

Hide and Show Information

25. Show all hidden rows and columns

When this macro is applied, the code will unhide all rows and columns in all sheets.

```
Sub UnhideAll()
Dim Ws As Worksheet
For Each Ws In Application.Worksheets
Ws.Select
Cells.Select
Selection.EntireRow.Hidden = False
Selection.EntireColumn.Hidden = False
Next
End Sub
```

26. Hide and Show all sheets

This routine hides and shows all sheets automatically. Once run the code, all worksheets will be hided except the first one, when run it again, the macro unhide all of them.

Obs. The macro starts from worksheet number 2, as the number in bold bellow. To start hide from another sheet, just change this number.

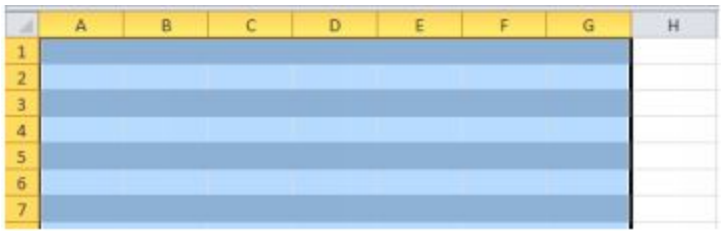
```
Sub HideAndShow()
For i = 2 To Worksheets.Count
   If Worksheets(i).Visible = True Then
      Worksheets(i).Visible = False
   Else
      Worksheets(i).Visible = True
   End If
Next
End Sub
```

Others

27. Apply alternate colors in a selection

The alternate colors can improve a lot the reading of information in a worksheet. The only way the Excel offer this resource without macros, is applying the table format, what's not always necessary or desired. This macro, applies the alternate colors easily.

Obs 1. Change the example values of color highlighted in bold
Obs 2. Just run the macro after select the area




```
Sub AlternateColors()
Dim SelecArea As Range
Dim LineVar As Range
Set SelecArea = Selection
For Each LineVar In SelecArea.Rows
   If LineVar.Row Mod 2 = 1 Then
      LineVar.Interior.ColorIndex = 15
   End If
Next LineVar
End Sub
```

28. Consolidate all worksheet's data's in the first one

Sometimes a file extracted from a database, have a lot of information fragmented in a lot of worksheets with the same format. This routine allows

users to easily compile all worksheet’s information in the first worksheet.

Obs 1. The code consolidates all data’s starting from A1 cell, what is changeable.
Obs 2. The first worksheet in the workbook will receive all data.

```
Sub Consolidate()
Dim total As Long
For i = 2 To Worksheets.Count
   Worksheets(i).Select
   Range("A1").Select
   If ActiveCell.Offset(0, 1) <> "" Then
      Range(Selection, Selection.End(xlToRight)).Select
   End If
   If ActiveCell.Offset(1, 0) <> "" Then
   Range(Selection, Selection.End(xlDown)).Select
   End If
Selection.Copy
Worksheets(1).Select
total =
Application.WorksheetFunction.CountA(Worksheets(1).Range("A1:A1048
576"))
Range("A" & total + 1).Select
ActiveSheet.Paste
Next
End Sub
```

29. Record time of other macros

Some macros are slow to process all algorithms, so it’s a good practice measure the time the code spent running to do adjust and improvements. This macro works as a stopwatch, what measure the amount of time that is necessary to execute another macro.

Obs. Change the example values highlighted in bold

```
Sub Runtime()
‘Start Timer
ti = Time
‘Inser the code here
‘End timer
tf = Time
‘Calculate the difference between start and end
tDif = tf - ti
MsgBox "Processing time: " & WorksheetFunction.text(tDif,
"HH:MM:SS")
End Sub
```

30. Copy and paste values in all worksheets

Sometimes, when a user need to save an Excel file, it’s not interesting show all formulas. This routine automates the process of copy and paste values in all worksheets.

```
Sub Copy_Paste_Values()
iquestion = MsgBox("This macro will convert the content off all cells to
values, continue?", vbYesNo)
If iquestion = vbYes Then
   For i = 1 To Worksheets.Count
      Worksheets(i).Select
      If Worksheets(i).Visible = False Then
         Worksheets(i).Visible = True
      End If
      Cells.Select
      Selection.Copy
      Selection.PasteSpecial Paste:=xlPasteValues
      Range("A1").Select
   Next
End If
End Sub
```

31. Remove empty spaces inside the cell

The goal of this code is remove the empty spaces before and after the cells content's, to run this routine, its necessary select a region and then run the macro.

```
Sub Remove_Empty()
Dim Celula As Range
For Each Celula In Selection
If Not IsEmpty(Celula) Then
Celula = Trim(Celula)
End If
Next Celula
End Sub
```

32. Refresh all Pivot Tables

This Macro is simple but useful, it just refresh all pivot tables in the active worksheet.

```
Sub Refresh_PivotTables()
  Dim pivotTable As PivotTable
  For Each pivotTable In ActiveSheet.PivotTables
    pivotTable.RefreshTable
  Next
End Sub
```

33. Remove duplicate values in each worksheet

This code is good for data treatment, its usual users who need to remove duplicate information in all sheets of some database extraction and this macro helps with this process.

Obs. The macro remove's duplicate in the same column in all worksheets, from row 1 to 99999

```
Sub Remove_Duplicate()
coluna = InputBox("Choose the column to remove duplicates in each
worksheet ?")
```

```
For i = 1 To Worksheets.Count
   Worksheets(i).Select
   ActiveSheet.Range("$" & coluna & "$1:$" & coluna &
"$99999").RemoveDuplicates Columns:=1, Header:=xlNo
Next
End Sub
```

34. Consolidate all worksheets in a new file

This code is simple, and it just consolidates the information off all worksheets in a single one, and then export those information's for a spreadsheet.

```
Sub ConsolidateNewFile()
Dim wkbDestino As String
Dim WorkbookName(1 To 99) As String
Dim ws As Worksheet
Dim i As Integer
i = 1
For Each Workbook In Workbooks
i = 1 + 1
WorkbookName(i) = Workbook.Name
Next Workbook
Total = i
Workbooks.Add
wkbDestiny = ActiveWorkbook.Name
For i = 1 To Total
   Workbooks(i).Activate
   For Each ws In Workbooks(i).Worksheets
   ws.Copy
after:=Workbooks(wkbDestiny).Sheets(Workbooks(wkbDestiny).Sheets.Co
unt)
   Next ws
Next i
Application.DisplayAlerts = False
```

```
For i = 1 To 3
Sheets(1).Delete
Next i
Application.DisplayAlerts = True
End Sub
```

35. Delete empty worksheet’s

This macro verifies all worksheets, looking for those what is empty of information, and then delete them.

```
Sub DeleteEmptySheet()
Dim Ws As Worksheet
On Error Resume Next
Application.DisplayAlerts = False
For Each Ws In Application.Worksheets
If Application.WorksheetFunction.CountA(Ws.UsedRange) = 0 Then
Ws.Delete
End If
Next
Application.DisplayAlerts = True
End Sub
```

36. Sort worksheets alphabetically

This macro sorts worksheets in alphabetically ascending order. To change to descending order, just change the signal (“>”), highlighted in bold, for the (“<”).

```
Sub OrderSheets()
Dim i As Integer
Dim j As Integer
Dim Resposta As VbMsgBoxResult
Resposta = MsgBox("Do you want order ascending ?", vbYesNo +
vbQuestion + vbDefaultButton1, "Order Alphabetically")
For i = 1 To Sheets.Count
For j = 1 To Sheets.Count - 1
```

```
If Resposta = vbYes Then
If UCase$(Sheets(j).Name) > UCase$(Sheets(j + 1).Name) Then
Sheets(j).Move After:=Sheets(j + 1)
End If
End If
Next j
Next i
End Sub
```

37. Change all worksheets name

This macro change all the worksheets name in a fast way with inputbox's

```
Sub ChangeSheetName()
For Each Sheet In Worksheets
Shortcut1:
Sheet.Select
NewName = InputBox("Qual o novo nome para esta Aba ?")
If NewName = "" Then
   Exit Sub
End If
On Error GoTo Shortcut1:
ActiveSheet.Name = NewName
Next
End Sub
```

38. Highlight row and column of a selected cell

This one is very good for user's who are used to frequently analyze a lot of information's. When this routine is running and the user select a cell, the row and column will highlight like in the image bellow:

Obs 1. It is necessary include the macro inside the specific worksheet object, in the example bellow, the code was added in the module worksheet "Sheet1".

	C	D	E	F	G
254					
255					
256	1	2	3	4	5
257	1	2	3	4	5
258	1	2	3	4	5
259	1	2	3	4	5
260	1	2	3	4	5
261	1	2	3	4	5
262	1	2	3	4	5
263	1	2	3	4	5
264	1	2	3	4	5
265	1	2	3	4	5

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
Dim LocalSelect As String
With Target
   If .Count = 1 Then
      LocalSelect = .Address & "," & .row & ":" & .row & "," & _
      Left(.Address, InStr(2, .Address, "$") - 1) & _
      ":" & Left(.Address, InStr(2, .Address, "$") - 1)
   End If
End With
On Error Resume Next
Range(LocalSelect).Select
End Sub
```

Interaction with Windows

39. Save selection range as image

This macro automatically save an Excel selection range, as an image in the same directory with the name of active sheet.

```
Sub SelectedRangeToImage()
   Dim iFilename As String
   Dim TempObjChart As Chart
   Dim Shp As Shape
   Dim Wsht As Worksheet
   Dim fileSaveName As Variant, pic As Variant
   Set Wsht = ActiveSheet
   Selection.Copy
   Wsht.Pictures.Paste.Select
   Set Shp = Wsht.Shapes(Wsht.Shapes.Count)
   Set TempObjChart = Charts.Add
   TempObjChart.ChartArea.Clear
   TempObjChart.Name = "PicChart" & (Rnd() * 10000)
   Set TempObjChart =
TempObjChart.Location(Where:=xlLocationAsObject,
Name:=Wsht.Name)
   TempObjChart.ChartArea.Width = Shp.Width
   TempObjChart.ChartArea.Height = Shp.Height
   TempObjChart.Parent.Border.LineStyle = 0
   Shp.Copy
   TempObjChart.ChartArea.Select
   TempObjChart.Paste
   iFilename = Application.ActiveWorkbook.Path & "\" &
ActiveSheet.Name & ".jpg"
   TempObjChart.Export Filename:=iFilename, FilterName:="jpg"
   Wsht.Cells(1, 1).Activate
   Wsht.ChartObjects(Wsht.ChartObjects.Count).Delete
   Shp.Delete
```

End Sub

40. Convert to PDF all spreadsheets inside a directory

This routine automatically converts all Excel files of an origin folder for PDF in a destiny a directory.

Obs. Change the example values highlighted in bold

```
Sub PdfConvert()
Dim iNumArq As Integer
Dim iCounter As Integer
Dim sMyFiles() As String
Dim OriginFolder As String
Dim DestinyFolder As String
OriginFolder = "C:\OriginFolder"
DestinyFolder = "C:\DestinyFolder"
FileFound = FindFiles(OriginFolder, sMyFiles, iNumArq, "*", True)
If FileFound Then
   For iCounter = 1 To iNumArq
   Filename = sMyFiles(2, iCounter)
   ExtCount = Len(Filename) - Application.WorksheetFunction.Search(".",
Filename, 1) + 1
   Workbooks.Open (OriginFolder & "\" & Filename)
   Workbooks(Filename).Activate
   ActiveSheet.ExportAsFixedFormat Type:=xlTypePDF, Filename:= _
      DestinyFolder & "\" & Mid(Filename, 1, Len(Filename) - ExtCount)
& ".pdf", Quality:= _
      xlQualityStandard, IncludeDocProperties:=True,
IgnorePrintAreas:=False, _
      OpenAfterPublish:=False
   Workbooks(Filename).Close
   Next iCounter
End If
```

```
End Sub
Function FindFiles(ByVal sPath As String, ByRef sFoundFiles() As String,
_
   ByRef iArqEncontrados As Integer, _
   Optional ByVal sFileSpec As String = "*.*", _
   Optional ByVal blIncludeSubFolders As Boolean = False) As Boolean
   Dim iCounter As Integer
   Dim sFileName As String
   Dim oFileSystem As Object, oParentFolder As Object, oFolder As
Object
   Set oFileSystem = CreateObject("Scripting.FileSystemObject")
   On Error Resume Next
   Set oParentFolder = oFileSystem.GetFolder(sPath)
   If oParentFolder Is Nothing Then
      FindFiles = False
      On Error GoTo 0
      Set oParentFolder = Nothing
      Set oFileSystem = Nothing
      Exit Function
   End If
   sPath = IIf(Right(sPath, 1) = "\", sPath, sPath & "\")
   sFileName = Dir(sPath & sFileSpec, vbNormal)
   Do While sFileName <> ""
      iCounter = UBound(sFoundFiles, 2)
      iCounter = iCounter + 1
      ReDim Preserve sFoundFiles(1 To 2, 1 To iCounter)
      sFoundFiles(1, iCounter) = sPath
      sFoundFiles(2, iCounter) = sFileName
      sFileName = Dir()
   Loop
   If blIncludeSubFolders Then
      For Each oFolder In oParentFolder.SubFolders
         FindFiles oFolder.Path, sFoundFiles, iArqEncontrados, sFileSpec,
blIncludeSubFolders
      Next
   End If
```

```
    FindFiles = UBound(sFoundFiles, 2) > 0
    iArqEncontrados = UBound(sFoundFiles, 2)
    On Error GoTo 0
    Set oFolder = Nothing
    Set oParentFolder = Nothing
    Set oFileSystem = Nothing
End Function
```

1. Find and list all files from a directory

This routine verifies a directory, finding for files and then lists all of them in Excel worksheet. This code is useful for users who constantly need to verify files available in a network.

Obs 1. Files will be listed in column A, starting by the first row.
Obs 2. Change the example values highlighted in bold

```
Sub ListFiles()
Dim iNumArq As Integer
 Dim iCounter As Integer
Dim sMyFiles() As String
Dim OriginFolder As String
 Dim DestinyFolder As String
 OriginFolder = "C:\OriginFolder"
Foundfile = FindFiles(OriginFolder, sMyFiles, iNumArq, "*", True)
If Foundfile Then
    For iCounter = 1 To iNumArq
    Filename = sMyFiles(2, iCounter)
    ActiveSheet.Range("A" & iCounter) = Filename
    Next iCounter
End If
Columns(1).AutoFit
End Sub
Function FindFiles(ByVal sPath As String, ByRef sFoundFiles() As String,
_
    ByRef iArqEncontrados As Integer, _
```

```
    Optional ByVal sFileSpec As String = "*.*", _
    Optional ByVal blIncludeSubFolders As Boolean = False) As Boolean
    Dim iCounter As Integer
    Dim sFileName As String
    Dim oFileSystem As Object, oParentFolder As Object, oFolder As
Object
    Set oFileSystem = CreateObject("Scripting.FileSystemObject")
    On Error Resume Next
    Set oParentFolder = oFileSystem.GetFolder(sPath)
    If oParentFolder Is Nothing Then
        FindFiles = False
        On Error GoTo 0
        Set oParentFolder = Nothing
        Set oFileSystem = Nothing
        Exit Function
    End If
    sPath = IIf(Right(sPath, 1) = "\", sPath, sPath & "\")
    sFileName = Dir(sPath & sFileSpec, vbNormal)
    Do While sFileName <> ""
        iCounter = UBound(sFoundFiles, 2)
        iCounter = iCounter + 1
        ReDim Preserve sFoundFiles(1 To 2, 1 To iCounter)
        sFoundFiles(1, iCounter) = sPath
        sFoundFiles(2, iCounter) = sFileName
        sFileName = Dir()
    Loop
    If blIncludeSubFolders Then
        For Each oFolder In oParentFolder.SubFolders
            FindFiles oFolder.Path, sFoundFiles, iArqEncontrados, sFileSpec,
blIncludeSubFolders
        Next
    End If
    FindFiles = UBound(sFoundFiles, 2) > 0
    iArqEncontrados = UBound(sFoundFiles, 2)
    On Error GoTo 0
    Set oFolder = Nothing
```

```
   Set oParentFolder = Nothing
   Set oFileSystem = Nothing
End Function
```

42. Copy any file from a directory to another

This macro is very good for persons who work with network, where many times are necessary copy a file from a path e paste in another directory. To help with this repetitive task, this code was made to do this work of copy and paste files from a directory to another

Obs. Change the example values highlighted in bold

Obs 2. The code can be improved with each user's necessities, by verifying the name of the file, or verifying part of the name with codes like MID(), LEFT() and other's codes that manipulates texts.

```
Sub Copyfiles()
Dim iNumArq As Integer
 Dim iCounter As Integer
Dim sMyFiles() As String
Dim OriginFolder As String
 Dim DestinyFolder As String
 OriginFolder = "C:\OriginFolder"
DestinyFolder = "C:\DestinyFolder"
FileFound = FindFiles(OriginFolder, sMyFiles, iNumArq, "*", True)
If FileFound Then
   For iCounter = 1 To iNumArq
   FileCopy OriginFolder & "\" & sMyFiles(2, iCounter), DestinyFolder &
"\" & sMyFiles(2, iCounter)
   Next iCounter
End If
End Sub
Function FindFiles(ByVal sPath As String, ByRef sFoundFiles() As String,
_
   ByRef iArqEncontrados As Integer, _
```

```
    Optional ByVal sFileSpec As String = "*.*", _
    Optional ByVal blIncludeSubFolders As Boolean = False) As Boolean
    Dim iCounter As Integer
    Dim sFileName As String
    Dim oFileSystem As Object, oParentFolder As Object, oFolder As
Object
    Set oFileSystem = CreateObject("Scripting.FileSystemObject")
    On Error Resume Next
    Set oParentFolder = oFileSystem.GetFolder(sPath)
    If oParentFolder Is Nothing Then
        FindFiles = False
        On Error GoTo 0
        Set oParentFolder = Nothing
        Set oFileSystem = Nothing
        Exit Function
    End If
    sPath = IIf(Right(sPath, 1) = "\", sPath, sPath & "\")
    sFileName = Dir(sPath & sFileSpec, vbNormal)
    Do While sFileName <> ""
        iCounter = UBound(sFoundFiles, 2)
        iCounter = iCounter + 1
        ReDim Preserve sFoundFiles(1 To 2, 1 To iCounter)
        sFoundFiles(1, iCounter) = sPath
        sFoundFiles(2, iCounter) = sFileName
        sFileName = Dir()
    Loop
    If blIncludeSubFolders Then
        For Each oFolder In oParentFolder.SubFolders
            FindFiles oFolder.Path, sFoundFiles, iArqEncontrados, sFileSpec,
blIncludeSubFolders
        Next
    End If
    FindFiles = UBound(sFoundFiles, 2) > 0
    iArqEncontrados = UBound(sFoundFiles, 2)
    On Error GoTo 0
    Set oFolder = Nothing
```

```
    Set oParentFolder = Nothing
    Set oFileSystem = Nothing
End Function
```

Interaction with Outolook

43. Send a simple e-mail with VBA

This macro sends a simple e-mail message with VBA code.

Obs 1. Change the subject and e-mail content message highlighted in bold
Obs 2. .CC and .BCC are optional; just use those lines for copies and blind copies if you need, else just erase them.
Obs 3. The **.Send** command sends the email automatically, if uses **.Display** instead, it create the e-mail but do not sends automatically.

```
Sub Send_Mail()
   Dim OutApp As Object
   Dim OutMail As Object
   Dim bMessage As String
   Set OutApp = CreateObject("Outlook.Application")
   Set OutMail = OutApp.CreateItem(0)
   bMessage = "Type the content line 1" & vbNewLine & _
            "Type the content line 2" & vbNewLine & _
            "Type the content line 3"
   On Error Resume Next
   With OutMail
      .to = "example@email.com"
      .CC = "copyemail@email.com"
      .BCC = "blindcopy@email.com"
      .Subject = "This is the Subject"
      .Body = "This is the email text"
      .Send
   End With
   On Error GoTo 0
   Set OutMail = Nothing
   Set OutApp = Nothing
End Sub
```

44. Send e-mail with workbook as attachment

This macro sends the last saved version of the active workbook in an outlook e-mail message.

Obs 1. Change the subject and e-mail message highlighted in bold
Obs 2. .CC and .BCC are optional; just use those lines for copies and blind copies
Obs 3. The **.Send** command sends the email automatically, if uses **.Display** instead, it create the e-mail but do not sends automatically.

```
Sub SendWorkbookEmail()
    Dim OutApp As Object
    Dim OutMail As Object
    Set OutApp = CreateObject("Outlook.Application")
    Set OutMail = OutApp.CreateItem(0)
    On Error Resume Next
    With OutMail
        .to = "example@email.com"
        .CC = "copyemail@email.com"
        .BCC = "blindcopy@email.com"
        .Subject = "This is the Subject"
        .Body = "This is the email text"
        .Attachments.Add ActiveWorkbook.FullName
        .Send
    End With
    On Error GoTo 0
    Set OutMail = Nothing
    Set OutApp = Nothing
End Sub
```

45. Send e-mail with active worksheet as attachment

This macro sends an e-mail using outlook with the active worksheet as attachment.

Obs 1. Change the subject and e-mail content highlighted in bold

Obs 2. .CC and .BCC are optional; just use those lines for copies and blind copies

Obs 3. The **.Send** command sends the email automatically, if uses **.Display** instead, it create the e-mail but do not sends automatically

```
Sub SendActiveSheetEmail ()
   Dim Exten As String
   Dim FormtN As Long
   Dim OutApp As Object
   Dim OutMail As Object
   Dim OriginWKB As Workbook
   Dim DestWKB As Workbook
   Dim TempFilePath As String
   Dim TempFileFolder As String
   Application.ScreenUpdating = False
   Application.EnableEvents = False
   Set OriginWKB = ActiveWorkbook
   ActiveSheet.Copy
   Set DestWKB = ActiveWorkbook
   With DestWKB
      If Val(Application.Version) < 12 Then
         Exten = ".xls": FormtN = -4143
      Else
         Select Case OriginWKB.FileFormat
         Case 51: Exten = ".xlsx": FormtN = 51
         Case 52:
            If .HasVBProject Then
```

```
                Exten = ".xlsm": FormtN = 52
            Else
                Exten = ".xlsx": FormtN = 51
            End If
        Case 56: Exten = ".xls": FormtN = 56
        Case Else: Exten = ".xlsb": FormtN = 50
        End Select
    End If
End With
TempFilePath = Environ$("temp") & "\"
TempFileFolder = "Part of " & OriginWKB.Name & " " & Format(Now,
"dd-mmm-yy h-mm-ss")
Set OutApp = CreateObject("Outlook.Application")
Set OutMail = OutApp.CreateItem(0)
With DestWKB
    .SaveAs TempFilePath & TempFileFolder & Exten,
FileFormat:=FormtN
    On Error Resume Next
    With OutMail
        .to = "example@email.com"
        .CC = "copyemail@email.com"
        .BCC = "blindcopy@email.com"
        .Subject = "This is the Subject"
        .Body = "This is the email text"
        .Attachments.Add DestWKB.FullName
        .Send
    End With
    On Error GoTo 0
    .Close savechanges:=False
End With
Kill TempFilePath & TempFileFolder & Exten
Set OutMail = Nothing
Set OutApp = Nothing
Application.ScreenUpdating = True
Application.EnableEvents = True
End Sub
```

46. Send e-mail with selection cells as attachment

This macro create a new file with the selection cells in Excel and send this new file as attachment through Outlook.

Obs 1. Change the subject and e-mail message highlighted in bold

Obs 2. .CC and .BCC are optional; just use those lines for copies and blind copies if you need, else just erase them
Obs 3. The **.Send** command sends the email automatically, if uses **.Display** instead, it create the e-mail but do not sends automatically

```
Sub Mail_Range()
   Dim wb As Workbook
   Dim iTempFolder As String
   Dim iTempFile As String
   Dim Source As Range
   Dim Dest As Workbook
   Dim iFormatNum As Long
   Dim iExt As String
   Dim OutApp As Object
   Dim OutMail As Object
   Set Source = Nothing
   On Error Resume Next
   Set Source = Selection.SpecialCells(xlCellTypeVisible)
   On Error GoTo 0
   If Source Is Nothing Then
      MsgBox "The source is out of range, please try again.", vbOKOnly
      Exit Sub
   End If
   With Application
      .ScreenUpdating = False
      .EnableEvents = False
   End With
```

```
    Set wb = ActiveWorkbook
    Set Dest = Workbooks.Add(xlWBATWorksheet)
    Source.Copy
    Dest.Sheets(1).Cells(1).PasteSpecial Paste:=8
    Dest.Sheets(1).Cells(1).PasteSpecial Paste:=xlPasteValues
    Dest.Sheets(1).Cells(1).PasteSpecial Paste:=xlPasteFormats
    Dest.Sheets(1).Cells(1).Select
    Application.CutCopyMode = False
    iTempFolder = Environ$("temp") & "\"
    iTempFile = "Selection of " & wb.Name & " " & Format(Now, "dd-
mmm-yy h-mm-ss")
    If Val(Application.Version) < 12 Then
       iExt = ".xls": iFormatNum = -4143
    Else
       iExt = ".xlsx": iFormatNum = 51
    End If
    Set OutApp = CreateObject("Outlook.Application")
    Set OutMail = OutApp.CreateItem(0)
    With Dest
       .SaveAs iTempFolder & iTempFile & iExt, FileFormat:=iFormatNum
       On Error Resume Next
       With OutMail
         .to = "example@email.com"
          .CC = "copyemail@email.com"
           .BCC = "blindcopy@email.com"
          .Subject = "This is the Subject"
           .Body = "This is the email text"
           .Attachments.Add Dest.FullName
           .Send
       End With
       On Error GoTo 0
       .Close savechanges:=False
    End With
    Kill iTempFolder & iTempFile & iExt
    Set OutMail = Nothing
    Set OutApp = Nothing
```

```
    With Application
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub
```

47. Send e-mail with other file as attachment

This macro sends automatically an e-mail with a file as attachment; it is possible combine this routine with the macro before, sending a file plus other file as attachment.

Obs 1. Change the subject and e-mail message highlighted in bold
Obs 2. .CC and .BCC are optional; just use those lines for copies and blind copies if you need, else just erase them
Obs 3. The **.Send** command sends the email automatically, if uses **.Display** instead, it create the e-mail but do not sends automatically

```
Sub SendWorkbookEmail()
    Dim OutApp As Object
    Dim OutMail As Object
    Set OutApp = CreateObject("Outlook.Application")
    Set OutMail = OutApp.CreateItem(0)
    On Error Resume Next
    With OutMail
        .to = "example@email.com"
        .CC = "copyemail@email.com"
        .BCC = "blindcopy@email.com"
        .Subject = "This is the Subject"
        .Body = "This is the email text"
        .Attachments.Add ("C:\examplefile.txt")
        .Send
    End With
    On Error GoTo 0
    Set OutMail = Nothing
    Set OutApp = Nothing
```

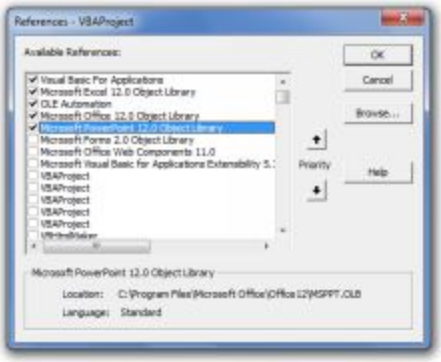
```
End Sub
```

Interaction with PowerPoint

48. Export charts for Microsoft PowerPoint presentation

This macro automatically creates a PowerPoint presentation with all charts objects inside the active worksheet.

Obs 1. It is possible to adjust the position and size of the chart changing bold highlighted text
Obs 2. Inside VBA editor, click in tools, reference and then scroll down to Microsoft PowerPoint object library, check the box and then press OK




```
Sub Excel_chart_to_PPT ()
    Dim PptApp As PowerPoint.Application
    Dim iSlide As PowerPoint.Slide
    Dim ChartObj As Excel.ChartObject
    On Error Resume Next
    Set PptApp = GetObject(, "PowerPoint.Application")
    On Error GoTo 0
    If PptApp Is Nothing Then
      Set PptApp = New PowerPoint.Application
    End If
    If PptApp.Presentations.Count = 0 Then
      PptApp.Presentations.Add
    End If
```

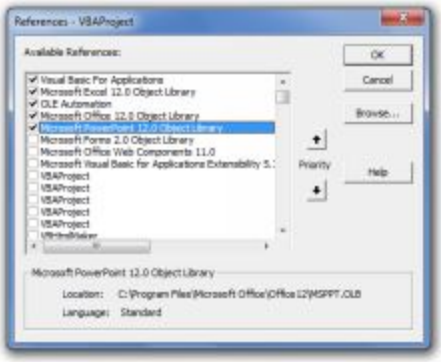
```
        PptApp.Visible = True
        For Each ChartObj In ActiveSheet.ChartObjects
            PptApp.ActivePresentation.Slides.Add
PptApp.ActivePresentation.Slides.Count + 1, ppLayoutText
            PptApp.ActiveWindow.View.GotoSlide
PptApp.ActivePresentation.Slides.Count
            Set iSlide =
PptApp.ActivePresentation.Slides(PptApp.ActivePresentation.Slides.Count
)
            ChartObj.Select
            ActiveChart.ChartArea.Copy
            On Error Resume Next

iSlide.Shapes.PasteSpecial(DataType:=ppPasteMetafilePicture).Select
            iSlide.Shapes(1).TextFrame.TextRange.Text =
ChartObj.Chart.ChartTitle.Text
            PptApp.ActiveWindow.Selection.ShapeRange.Left = 25
            PptApp.ActiveWindow.Selection.ShapeRange.Top = 150
            iSlide.Shapes(2).Width = 300
            iSlide.Shapes(2).Left = 600
        Next
    AppActivate ("Microsoft PowerPoint")
    Set iSlide = Nothing
    Set PptApp = Nothing
End Sub
```

49. Export Selection Range for Microsoft PowerPoint

This macro automatically creates a PowerPoint presentation with all charts objects inside active worksheet.

Obs 1. It is possible to adjust the position changing the highlight bold text
Obs 2. Inside VBA editor, click in tools, reference and then scroll down to Microsoft PowerPoint object library, check the box and then press OK




```
Sub Selection_to_PowerPoint()
Dim iRange As Range
Dim PptObj As Object
Dim iPresent As Object
Dim iSlide As Object
Dim iShape As Object
  Set iRange = Selection
  On Error Resume Next
      Set PptObj = GetObject(class:="PowerPoint.Application")
      Err.Clear
      If PptObj Is Nothing Then Set PptObj =
CreateObject(class:="PowerPoint.Application")
      If Err.Number = 429 Then
        MsgBox "PowerPoint could not be found, aborting."
        Exit Sub
      End If
  On Error GoTo 0
  Application.ScreenUpdating = False
  Set iPresent = PptObj.Presentations.Add
  Set iSlide = iPresent.Slides.Add(1, 11)
  iRange.Copy
  iSlide.Shapes.PasteSpecial DataType:=2
  Set iShape = iSlide.Shapes(iSlide.Shapes.Count)
      iShape.Left = 100
      iShape.Top = 160
  PptObj.Visible = True
  PptObj.Activate
```

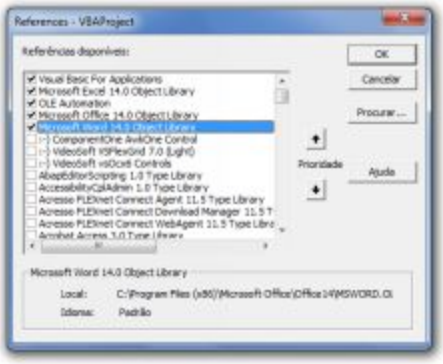
```
  Application.CutCopyMode = False
  Application.ScreenUpdating = True
End Sub
```

Interaction with Word

50. Export selection to Microsoft Word

This macro automatically export the current selection range in Excel for Microsoft Word.

Obs. Inside VBA editor, click in tools, reference and then scroll down to Microsoft Word object library, check the box and then press OK

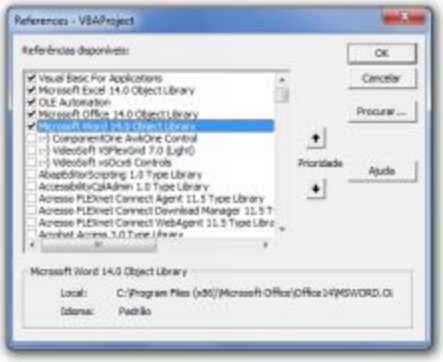



```
Sub CopyRangetoWord()
Dim WodAPP As Word.Application
Dim WordDOC As Word.Document
If Not TypeName(Selection) = "Range" Then
   MsgBox "Out of range, please try again.", vbExclamation, "Range Error"
Else
   Set WodAPP = GetObject(, "Word.Application")
   Set WordDOC = WodAPP.ActiveDocument
   Selection.Copy
   WodAPP.Selection.PasteSpecial Link:=False, DataType:=wdPasteRTF, _
      Placement:=wdInLine, DisplayAsIcon:=False
   Set WordDOC = Nothing
   Set WodAPP = Nothing
End If
End Sub
```

51. Export Active Sheet to Microsoft Word

This routine export the information of used range to a new file in Microsoft Word and save the file with the same name and the same directory.

Obs. Inside VBA editor, click in tools, reference and then scroll down to Microsoft Word object library, check the box and then press OK




```
Sub Worksheet_to_Word()
   Dim Question As Integer
   Question = MsgBox("This macro export all used range, do you want to
continue ?", vbYesNo)
   If question <> 6 Then
   Exit Sub
   End If
   Set Object = CreateObject("Word.Application")
   Object.Visible = True
   Set newObject = Object.Documents.Add
   ActiveSheet.UsedRange.Copy
   newObject.Range.Paste
   Application.CutCopyMode = False
   Object.Activate
   On Error Resume Next
   newObject.SaveAs Filename:=Application.ActiveWorkbook.Path & "\"
& ActiveSheet.Name
End Sub
```