VBA Utility User’s Guide

The primary functionality of the VBA Utility is to browse Visual Basic for Applications (VBA) code in Word and Excel documents. The application provides access to the VB integrated development environment (VBIDE) of both Word and Excel, as well as, exported .BAS files. The ability to view exported .BAS files allows access to VBA originating in any application, including MS Outlook.

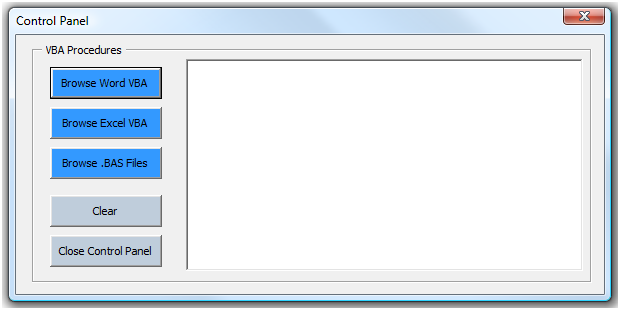
The utility is a custom, office automation, application using MS Word and MS Excel to browse, to select and to view VBA code modules in documents (before exporting the code).

# Design Considerations.

Each Word or Excel document contains its own VB Project. Several Word documents can be opened at the same time with all VBA projects showing in one instance of the VBIDE. The same is true for Excel. However, Word and Excel VBIDEs are separate. Therefore, a Word VBIDE must be opened for Word documents and an Excel VBIDE must be opened for Excel documents (Outlook is discussed later).

This VBA Utility uses MS Word as the user interface. Excel applications are opened when needed to access the Excel IDE, but the user only interacts with Word. (The Excel VBA Utility is opened with visibility set to false.)

Note: The user could choose to open the Excel VBA Utility directly; however, a simpler interface with more functionality is provided using the automation approach.

The Control Panel, userform takes the user’s input to browse for files, then to select code modules. The buttons provide the functionality indicated by their captions. The three browse buttons each opens a browse [filedialog](#FileDialog), filtered for the indicated file type.

The ‘Browse Excel VBA’ opens the Excel VBA Utility ([Excel.Application.Visible=False](#VisibleFalse)). The ‘Clear’ button clears the listbox. The ‘Close Control Panel’, as expected, closes the Control Panel.

With OWord.FileDialog(**msoFileDialogOpen**)

.InitialFileName = newPath

.Filters.Clear

.Filters.Add "Word files", "\*.do\*, filetype"

.Title = "Select File"

.AllowMultiSelect = False

End With

On Error Resume Next

Set OExcel = GetObject(, "Excel.Application")

If Err Then

Set OExcel = New Excel.Application

End If

workbookPath = ThisDocument.path & "\VBAUtility.xlsm"

Set OWB = OExcel.Workbooks.Open(workbookPath)

**OExcel.Visible = False**

The listbox is populated by the procedures found in the browsed file. The click event on an items in the list triggers a procedure that loads the selected module in a new Word document. The selected procedure name is found, and the document is formatted to make it easier to read. The procedures in a module are found programmatically by stepping through each line of code in the module, while using the .[ProcOfLine()](#ProcOfLine) property to find the procedure name (Microsoft Visual Basic for Applications Extensibility 5.3 library is set in the VBIDE References).

For I = .CountOfDeclarationLines + 1 To .CountOfLines

If Trim(.Lines(I, 1)) <> "" Then

If strTemp <> .Lines \_

(.ProcBodyLine(.**ProcOfLine**(I, \_

vbext\_pk\_Proc), vbext\_pk\_Proc), 1) Then

countProcedures = countProcedures + 1

totalProcedures = totalProcedures + 1

strTemp = .Lines(.ProcBodyLine(.ProcOfLine(I, \_

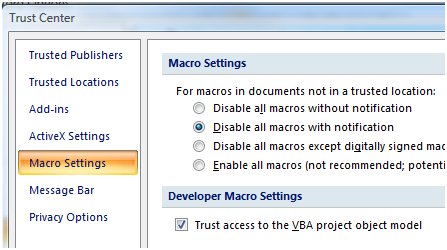
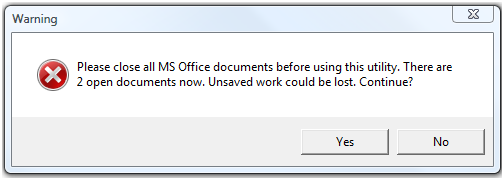
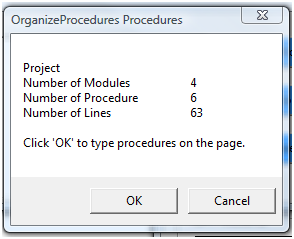
vbext\_pk\_Proc), vbext\_pk\_Proc), 1)

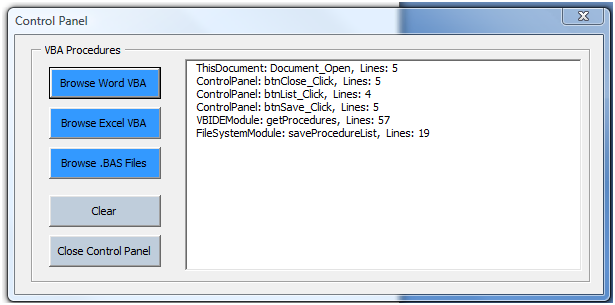
End If

End If

Next I

# Step-By-Step Guide to Using the VBA Utility

1. Download both the Word and Excel parts of the VBAUtility and save them in a Trusted Location (Trusted Locations are set in the Trust Center, 1). In the Trust Center make sure that ‘Trust access to the VBA project object model ‘ is checked (2).
2. To open the utility, double-click on VBAUtility.docm where it is saved. This opens the file in MS Word. A warning pops-up. The best practice is to close all other Word documents when using this utility. Unsaved work could be lost as several files are opened and closed using automation techniques. This window only pops-up when Documents.Count <> 1. Click-on ‘Yes’ to continue; click-on ‘No’ to close the utility.
3. The Control Panel shows in the lower right corner of the screen (after ‘Yes’ is clicked).
4. Click-on the ‘Browse Word VBA’ button to open the browse dialog and select a document. This will open that documents and run a procedure that counts the number of modules, number of procedures and number of code lines in the documents. A window pops-up that displays this information. Optionally, click-on ‘OK’, to programmatically type the list onto the page or click-on ‘Cancel’ to continue.
5. The listbox is populated with the Module Names, Procedure Names, and Number of Code Lines in each procedure found in the document.



1. Click-on a procedure in the list (1) to view the complete code in the module containing the selected procedure. In this image the list contains 6 procedures. The ControlPanel module contains three procedures of the six. There are three other modules shown, each contains one procedure.
2. A Word document is populated with the selected module. Below is shown the output when one of the listed procedures is selected. This module is the code behind the Control Panel in the ‘OrgnaizeProcedures’ documents (not the VBA Utility). The Control Panel attributes are shown, as well as, the procedures associated with the buttons.

VERSION 5.00

Begin {C62A69F0-16DC-11CE-9E98-00AA00574A4F} ControlPanel

Caption = "Control Panel"

ClientHeight = 2865

ClientLeft = 45

ClientTop = 345

ClientWidth = 6720

OleObjectBlob = "ControlPanel.frx":0000

StartUpPosition = 1 'CenterOwner

End

Attribute VB\_Name = "ControlPanel"

Attribute VB\_GlobalNameSpace = False

Attribute VB\_Creatable = False

Attribute VB\_PredeclaredId = True

Attribute VB\_Exposed = False

Option Explicit

**Private Sub btnClose\_Click()**

Unload Me

End Sub

**Private Sub btnList\_Click()**

VBIDEModule.getProcedures

End Sub

**Private Sub btnSave\_Click()**

FileSystemModule.saveProcedureList

End Sub

1. The other two browse buttons trigger similar functionality, but the nature of the files requires distinctly different techniques to implement the functionality. Click-on ‘Browse Excel VBA’ to open the ExcelVBAUtility.xlsm. The Excel application opens with the ExcelVBAUtility workbook, but is not visible. A browse dialog opens to select an Excel workbook. The VBA procedures in the selected workbook project populate the listbox and populate Sheet1 of the utility workbook.
2. As with the Word procedures list, click-on any procedure in the list to view the entire code module containing that procedure.

Note: The tricky part of this functionality is transferring the procedure names found with the Excel VBIDE to the Word listbox, then accessing the Excel VBIDE to view the selected code module in a Word document. The listbox is populated, as shown in the code snippet, by counting the procedures, then populating an array in Word with the data from the named range, ‘proceduresTable’. This type of automation is accomplished by adding References in Word to the Excel Library of procedures.

'run function in Excel to count procedures

proceduresCount = Run("AutomationModule.proceduresCount")

FileToGet = Range("A1").Value

'dimension variables with count

ReDim dataArray(proceduresCount, 2)

Range("A3:B" & proceduresCount + 2).Name = "proceduresTable"

dataArray = Range("proceduresTable")

For I = 1 To UBound(dataArray)

If dataArray(I, 1) <> vbNullString Then

ControlPanel.ProcedureListBox.AddItem dataArray(I, 1) & ", Lines: " & dataArray(I, 2)

End If

Next I

1. The third browse button, ‘Browse .BAS Files’, opens a browse dialog, filtered for .BAS files. The selected .BAS file is opened in Word and formatted for easier reading.
2. Click-on the ‘Clear’ button to close all instances of Word (except the utility), to clear the listbox and to clear the page.
3. Click-on the ‘Close Control Panel’ button to unload the Control Panel userform.
4. Click-on the red-x in the upper right corner of the window to close the utility. This action triggers a procedure that closes all open documents, hidden or otherwise.

**Caution: Unsaved work could be lost.**