ProjectManager.xlsm

www.github.com/alexofrhodes

--- Table Of Contents ---

--- ThisWorkbook ---

--- ProjectManagerSettings - Sheet11 ---

--- ProjectManagerPrinter - Sheet9 ---

--- ProjectManagerTXTColour - Sheet12 ---

--- Userforms - Sheet25 ---

```
Option Explicit
```

--- README - Sheet22 ---

--- clsEditOpenXML ---

```
Option Explicit
Private mbCreateBackup As Boolean
Private mvSourceFile As Variant
Private msSheet2Change As String
Private msSheetId As String
Private msSheetFileName As String
Private mbAddedZip As Boolean
Private mvXMLFolderRoot As Variant
Private mvxmlfolder As Variant
'Private mvXMLFolderCustomUI As Variant
Public Enum XMLFolder
    'Date Created : 5/12/2009 21:34
    'Author       : Ken Puls (www.excelguru.ca)
    'Macro Purpose: Constants for XML Containers
    XMLFolder_root = 1
    XMLFolder_rels = 2
    XMLFolder_xl = 3
    XMLFolder_customUI = 4
    XMLFolder_docProps = 5
End Enum

Public Function GetXMLFromFile(sFileName As String, sXMLFolder As  _
XMLFolder) As String
    '---------------------------------
    '-----------------------------------------
    ' Procedure : GetXMLFromFile
    ' Company   : JKP Application Development Services (c)
    ' Author    : Jan Karel Pieterse
    ' Created   : 6-5-2009
    ' Purpose   : Gets the XML code from the foldername\filename
    '---------------------------------
    '-----------------------------------------
    Dim oXMLDoc As MSXML2.DOMDocument
    If Len(XMLFolder(sXMLFolder)) = 0 Then
        GetXMLFromFile = ""
    Else
        Set oXMLDoc = New MSXML2.DOMDocument
        oXMLDoc.Load XMLFolder(sXMLFolder) & sFileName
        GetXMLFromFile = oXMLDoc.XML
        Set oXMLDoc = Nothing
    End If
End Function

Public Sub WriteXML2File(sXML As String, sFileName As String, sXMLFolder  _
As XMLFolder)
    '---------------------------------
    '-----------------------------------------
    ' Procedure : WriteXML2File
    ' Company   : JKP Application Development Services (c)
    ' Author    : Jan Karel Pieterse
    ' Created   : 6-5-2009
```

```
    ' Purpose   : Writes sXML to sFileName
    '             Modified by Ken Puls 2009-05-12
    '             Adjusted to add ability to write to customUI container
    '---------------------------------
    '-------------------------------------
    Dim oXMLDoc As MSXML2.DOMDocument
    Set oXMLDoc = New MSXML2.DOMDocument
    'If attempting to write a customU
    'I component, test to see if one exists
    'Should probably test the .rels file to
    'see if the CustomUI relationship exists...
    If sXMLFolder = XMLFolder_customUI Then
        If Not FolderExists(XMLFolder(XMLFolder_customUI)) Then
            MkDir XMLFolder(XMLFolder_customUI)
            'Write the XML to the file
            oXMLDoc.loadXML sXML
            oXMLDoc.Save XMLFolder(sXMLFolder) & sFileName
            'CustomUI has not been crea
            'ted yet.  Rels file needs to be adjusted
            AddCustomUIToRels
        End If
    End If
    'Write the XML to the file
    oXMLDoc.loadXML sXML
    oXMLDoc.Save XMLFolder(sXMLFolder) & sFileName
End Sub

Public Sub AddCustomUIToRels()
    'Date Created : 5/14/2009 23:29
    'Author       : Ken Puls (www.excelguru.ca)
    'Macro Purpose: Add the customUI relationship to the rels file
    Dim oXMLDoc As MSXML2.DOMDocument
    '    Dim oXMLElement As MSXML2.IXMLDOMElement
    Dim oXMLElement As MSXML2.IXMLDOMNode
    Dim oXMLAttrib As MSXML2.IXMLDOMAttribute
    Dim oNamedNodeMap As MSXML2.IXMLDOMNamedNodeMap
    Dim oXMLRelsList As MSXML2.IXMLDOMNodeList
    'Create a new XML document
    Set oXMLDoc = New MSXML2.DOMDocument
    'Attach to the root element of the .rels file
    oXMLDoc.Load XMLFolder(XMLFolder_rels) & ".rels"
    'Create a new relationship element in the .rels file
    Set oXMLElement = oXMLDoc.createNode(1, "Relationship", "http:// _
    schemas.openxmlformats.org/package/2006/relationships")
    Set oNamedNodeMap = oXMLElement.Attributes
    'Create ID attribute for the element
    Set oXMLAttrib = oXMLDoc.createAttribute("Id")
    oXMLAttrib.nodeValue = "cuID"
    oNamedNodeMap.setNamedItem oXMLAttrib
    'Create Type attribute for the element
    '    Set oXMLAttrib = oXMLDoc.createAttribute("Type")
    '    oXMLAttrib.nodeValue = "http://shemas.micr
```

```
        'osoft.com/office/2006/relationships/ui/extensibility"
        Set oXMLAttrib = oXMLDoc.createAttribute("Type")
        oXMLAttrib.nodeValue = "http://schemas.microsoft.com/office/2006/ _
        relationships/ui/extensibility"
        oNamedNodeMap.setNamedItem oXMLAttrib
        'Create Target element for the attribute
        '    Set oXMLAttrib = oXMLDoc.createAttribute("Target")
        '    oXMLAttrib.nodeValue = "customUI/customUI.xml"
        '    oXMLElement.setAttributeNode oXMLAttrib
        Set oXMLAttrib = oXMLDoc.createAttribute("Target")
        oXMLAttrib.nodeValue = "customUI/customUI.xml"
        oNamedNodeMap.setNamedItem oXMLAttrib
        'Now insert the new node at the proper location
        Set oXMLRelsList = oXMLDoc.selectNodes("/Relationships")
        oXMLRelsList.Item(0).appendChild oXMLElement
        'Save the .rels file
        oXMLDoc.Save XMLFolder(XMLFolder_rels) & ".rels"
        Set oXMLAttrib = Nothing
        Set oXMLElement = Nothing
        Set oXMLDoc = Nothing
    End Sub

    Private Function GetSheetIdFromSheetName(sSheetName) As String
        '---------------------------------
        '-------------------------------------
        ' Procedure : GetSheetIdFromSheetName
        ' Company   : JKP Application Development Services (c)
        ' Author    : Jan Karel Pieterse
        ' Created   : 6-5-2009
        ' Purpose   : Finds out what the SheetId of sSheetname is
        '             by reading Workbook.xml
        '---------------------------------
        '-------------------------------------
        Dim oXMLDoc As MSXML2.DOMDocument
        Dim oxmlNode As MSXML2.IXMLDOMNode
        Dim oXMLChildNode As MSXML2.IXMLDOMNode
        Dim oXMLTemp As MSXML2.IXMLDOMNode
        If XMLFolder(XMLFolder_xl) <> "" And Sheet2Change <> "" Then
            Set oXMLDoc = New MSXML2.DOMDocument
            oXMLDoc.Load XMLFolder(XMLFolder_xl) & "workbook.xml"
            For Each oxmlNode In oXMLDoc.ChildNodes
                For Each oXMLChildNode In oxmlNode.ChildNodes
                    If oXMLChildNode.baseName = "sheets" Then
                        For Each oXMLTemp In oXMLChildNode.ChildNodes
                            If oXMLTemp.Attributes.getNamedItem("name"). _
                            nodeValue = sSheetName Then
                                GetSheetIdFromSheetName = oXMLTemp.Attributes. _
                                getNamedItem("r:id").nodeValue
                                Exit Function
                            End If
                        Next
                    End If
```

```
            Next
        Next
    End If
End Function

Public Function GetSheetFileNameFromId(sSheetId As String) As String
    '---------------------------------
    '-------------------------------------
    ' Procedure : GetSheetFileNameFromId
    ' Company   : JKP Application Development Services (c)
    ' Author    : Jan Karel Pieterse
    ' Created   : 6-5-2009
    ' Purpose   : Fetches the name of the xml
    'file belonging to the sheet with id SheetId.
    '---------------------------------
    '-------------------------------------
    Dim oXMLDoc As MSXML2.DOMDocument
    Dim oxmlNode As MSXML2.IXMLDOMNode
    Dim oXMLChildNode As MSXML2.IXMLDOMNode
    If XMLFolder(XMLFolder_xl) <> "" And Sheet2Change <> "" Then
        Set oXMLDoc = New MSXML2.DOMDocument
        oXMLDoc.Load XMLFolder(XMLFolder_xl) & "_rels\workbook.xml.rels"
        For Each oxmlNode In oXMLDoc.ChildNodes
            For Each oXMLChildNode In oxmlNode.ChildNodes
                If oXMLChildNode.Attributes.getNamedItem("Id").nodeValue  _
                = sSheetId Then
                    GetSheetFileNameFromId = oXMLChildNode.Attributes. _
                    getNamedItem("Target").nodeValue
                    Exit Function
                End If
            Next
        Next
    End If
End Function

Private Function GetSheetNameFromId(sId As String) As String
    '---------------------------------
    '-------------------------------------
    ' Procedure : GetSheetNameFromId
    ' Company   : JKP Application Development Services (c)
    ' Author    : Jan Karel Pieterse
    ' Created   : 6-5-2009
    ' Purpose   : Returns the sheetname belonging to a sheetId
    '---------------------------------
    '-------------------------------------
    Dim oXMLDoc As MSXML2.DOMDocument
    Dim oxmlNode As MSXML2.IXMLDOMNode
    Dim oXMLChildNode As MSXML2.IXMLDOMNode
    Dim oXMLChildChildNode As MSXML2.IXMLDOMNode
    If mvxmlfolder(XMLFolder_xl) <> "" Then
        Set oXMLDoc = New MSXML2.DOMDocument
        oXMLDoc.Load XMLFolder(XMLFolder_xl) & "workbook.xml"
```

```
        For Each oxmlNode In oXMLDoc.ChildNodes
            For Each oXMLChildNode In oxmlNode.ChildNodes
                If oXMLChildNode.nodeName = "sheets" Then
                    For Each oXMLChildChildNode In oXMLChildNode. _
                    ChildNodes
                        If oXMLChildChildNode.Attributes. _
                        getNamedItem("r:id").nodeValue = "rId" & val(sId)  _
                        + 1 Then
                            GetSheetNameFromId = oXMLChildChildNode. _
                            Attributes.getNamedItem("name").nodeValue
                            'Got it, get out
                            Exit Function
                        End If
                    Next
                    'get out here, no point in doing the rest
                    Exit Function
                End If
            Next
        Next
    End If
End Function

Public Sub ZipAllFilesInFolder()
    '---------------------------------
    '---------------------------------------
    ' Procedure : ZipAllFilesInFolder
    ' Company   : JKP Application Development Services (c)
    ' Author    : Jan Karel Pieterse
    ' Created   : 6-5-2009
    ' Purpose   : Zips all files in a folder (includ
    'ing subfolders) whilst retaining the folder structure
    '---------------------------------
    '---------------------------------------
    'Courtesy www.rondebruin.nl
    Dim oShellApp As Object
    Dim sDate As String
    Dim sDefPath As String
    Dim vFileNameZip As Variant
    Dim fso As Object
    Dim lFileCt As Long
    Set fso = CreateObject("scripting.filesystemobject")
    'To ensure a unique filename,
    'append date and time to the name of the current file
    sDate = Format(Now, " dd-mmm-yy h-mm-ss")
    vFileNameZip = SourceFile & sDate & ".zip"
    'Create empty Zip File
    NewZip vFileNameZip
    Set oShellApp = CreateObject("Shell.Application")
    'Count how many items are in the "old" folder
    lFileCt = oShellApp.Namespace(FolderName & "Unzipped " & FileName &  _
    Application.PathSeparator).items.Count
    'Copy the files to the compressed folder
```

```
        oShellApp.Namespace(vFileNameZip).copyhere oShellApp. _
        Namespace(FolderName & "Unzipped " & FileName & Application. _
        PathSeparator).items
        'Keep script waiting until we have same # of files in the new folder
        On Error Resume Next
        Do Until oShellApp.Namespace(vFileNameZip).items.Count = lFileCt
            Application.Wait (Now + TimeValue("0:00:01"))
        Loop
        DoEvents
        'Remove original file
        Kill SourceFile
        'Rename new zipped file to same nam
        'e as original file (with .zip appended)
        Name vFileNameZip As SourceFile
        On Error Resume Next
        'Now remove old folder, just in case something went haywire
        fso.DeleteFolder FolderName & "Unzipped " & FileName, True
        On Error GoTo 0
        Set oShellApp = Nothing
    End Sub

    Public Sub UnzipFile()
        '---------------------------------
        '-------------------------------------
        ' Procedure : UnzipFile
        ' Company   : JKP Application Development Services (c)
        ' Author    : Jan Karel Pieterse
        ' Created   : 6-5-2009
        ' Purpose   : Unzips all files in a zip file to a designated folder
        '             Modified by Ken Puls 2009-05-12
        '             Adjusted to record customUI portion
        '---------------------------------
        '-------------------------------------
        'Courtesy www.rondebruin.nl
        Dim fso As Object
        Dim oShellApp As Object
        Set fso = CreateObject("scripting.filesystemobject")
        'Derive the folder to unzip to from the location of the sourcefile
        XMLFolderRoot = FolderName
        'A dedicated unzip folder will be cre
        'ated in the same folder as the sourcefile,
        'called ..\Unzipped Filename\
        If Right(XMLFolderRoot, 1) <> Application.PathSeparator Then
            XMLFolderRoot = XMLFolderRoot & "\UnZipped " & FileName &  _
            Application.PathSeparator
        Else
            XMLFolderRoot = XMLFolderRoot & "UnZipped " & FileName &  _
            Application.PathSeparator
        End If
        On Error Resume Next
        'Remove all previous existing folders
        fso.DeleteFolder XMLFolderRoot & "*", True
```

```
        Kill XMLFolderRoot & "*.*"
        'Create normal folder
        If FolderExists(XMLFolderRoot) = False Then
            MkDir XMLFolderRoot
        End If
        Set oShellApp = CreateObject("Shell.Application")
        'Copy the files in the newly created folder
        oShellApp.Namespace(XMLFolderRoot).copyhere oShellApp. _
        Namespace(SourceFile).items
        On Error Resume Next
        'Clean up temp folder
        fso.DeleteFolder Environ("Temp") & "\Temporary Directory*", True
        Set oShellApp = Nothing
        Set fso = Nothing
        Exit Sub
    End Sub

    Sub NewZip(sPath)
        'Courtesy www.rondebruin.nl
        'Create empty Zip File
        'Changed by keepITcool Dec-12-2005
        If Len(DIR(sPath)) > 0 Then Kill sPath
        Open sPath For Output As #1
        Print #1, Chr$(80) & Chr$(75) & Chr$(5) & Chr$(6) & String(18, 0)
        Close #1
    End Sub

    Public Property Get CreateBackup() As Boolean
        CreateBackup = mbCreateBackup
    End Property

    Public Property Let CreateBackup(ByVal bCreateBackup As Boolean)
        mbCreateBackup = bCreateBackup
    End Property

    Private Sub Class_Initialize()
        'Set defaults
        CreateBackup = True
    End Sub

    Public Property Get SourceFile() As Variant
        SourceFile = mvSourceFile
    End Property

    Public Property Let SourceFile(ByVal vSourceFile As Variant)
        mvSourceFile = vSourceFile
        If CreateBackup Then
            If Len(DIR(vSourceFile & "(backup)")) > 0 Then
                Kill vSourceFile & "(backup)"
            End If
            FileCopy vSourceFile, vSourceFile & "(backup)"
        End If
```

```
        If Not vSourceFile Like "*.zip" Then
            Name vSourceFile As vSourceFile & ".zip"
            mvSourceFile = mvSourceFile & ".zip"
            AddedZip = True
        End If
    End Property

    Public Property Get FolderName() As Variant
        FolderName = Mid(SourceFile, 1, InStrRev(SourceFile, Application. _
        PathSeparator))
    End Property

    Public Property Get FileName() As Variant
        If SourceFile <> "" Then
            FileName = Mid(SourceFile, InStrRev(SourceFile, Application. _
            PathSeparator) + 1, Len(SourceFile))
        End If
    End Property

    'Public Property Get xmlfolder(XMLFolder_xl)() As Variant
    '    xmlfolder(XMLFolder_xl) = mvxmlfolder(XMLFolder_xl)
    'End Property
    '
    'Public Property Let xmlfolder(XMLFolder_xl
    ')(ByVal vxmlfolder(XMLFolder_xl) As Variant)
    '    mvxmlfolder(XMLFolder_xl) = vxmlfolder(XMLFolder_xl)
    'End Property
    '
    'Public Property Get XMLFolderCustomUI() As Variant
    ''Date Created : 5/12/2009 21:18
    ''Author       : Ken Puls (www.excelguru.ca)
    ''Macro Purpose: Retrieve customUI folder
    '    XMLFolderCustomUI = mvXMLFolderCustomUI
    'End Property
    '
    'Public Property Let XMLFolderCustomU
    'I(ByVal vXMLFolderCustomUI As Variant)
    ''Date Created : 5/12/2009 21:18
    ''Author       : Ken Puls (www.excelguru.ca)
    ''Macro Purpose: Save customUI folder
    '    mvXMLFolderCustomUI = vXMLFolderCustomUI
    'End Property
    Public Property Get XMLFolder(sXMLFolder As XMLFolder) As String
        Select Case sXMLFolder
            Case Is = XMLFolder_root
                XMLFolder = mvXMLFolderRoot
            Case Is = XMLFolder_customUI
                XMLFolder = mvXMLFolderRoot & "customUI" & Application. _
                PathSeparator
            Case Is = XMLFolder_docProps
                XMLFolder = mvXMLFolderRoot & "docProps" & Application. _
                PathSeparator
```

```
        Case Is = XMLFolder_rels
            XMLFolder = mvXMLFolderRoot & "_rels" & Application. _
            PathSeparator
        Case Is = XMLFolder_xl
            XMLFolder = mvXMLFolderRoot & "xl" & Application.PathSeparator
    End Select
End Property

Public Property Get Sheet2Change() As String
    Sheet2Change = msSheet2Change
End Property

Public Property Let Sheet2Change(ByVal sSheet2Change As String)
    msSheet2Change = sSheet2Change
    SheetId = GetSheetIdFromSheetName(sSheet2Change)
    If SheetId <> "" Then
        SheetFileName = GetSheetFileNameFromId(SheetId)
    End If
End Property

Public Property Get SheetId() As String
    SheetId = msSheetId
End Property

Public Property Let SheetId(ByVal sSheetId As String)
    msSheetId = sSheetId
End Property

Public Property Get SheetFileName() As String
    SheetFileName = msSheetFileName
End Property

Public Property Let SheetFileName(ByVal sSheetFileName As String)
    msSheetFileName = sSheetFileName
End Property

Private Property Get AddedZip() As Boolean
    AddedZip = mbAddedZip
End Property

Private Property Let AddedZip(ByVal bAddedZip As Boolean)
    mbAddedZip = bAddedZip
End Property

Private Sub Class_Terminate()
    Dim fso As Object
    If AddedZip Then
        'Remove .zip from file again
        Name SourceFile As Left(SourceFile, Len(SourceFile) - 4)
    End If
    'Remove zip folder
    On Error Resume Next
```

```
        fso.DeleteFolder XMLFolderRoot, True
    End Sub

    Private Property Get XMLFolderRoot() As Variant
        XMLFolderRoot = mvXMLFolderRoot
    End Property

    Private Property Let XMLFolderRoot(ByVal vXMLFolderRoot As Variant)
        mvXMLFolderRoot = vXMLFolderRoot
    End Property

```

--- Home ---

```
Sub Main()
    uProjectManager.Show
End Sub

'Callback for buttonProManager onAction
Sub ProjectManagerButtonClicked(control As IRibbonControl)
    Main
End Sub

```

--- mProjectManager ---

```
Public pmWorkbook As Workbook
#If VBA7 Then
    Declare PtrSafe Sub keybd_event Lib "User32" (ByVal bVk As Byte, _
    ByVal bScan As Byte, ByVal dwFlags As Long, ByVal dwExtraInfo As  _
    LongPtr)
#Else
    Declare Sub keybd_event Lib "user32" (ByVal bVk As Byte, _
    ByVal bScan As Byte, ByVal dwFlags As Long, ByVal dwExtraInfo As Long)
#End If

Public Sub ExtractRibbonX(sFullFile As String, sSaveFile As String)
    '---------------------------------
    '-------------------------------------
    ' Procedure : Demo
    ' Company   : JKP Application Development Services (c)
    ' Author    : Jan Karel Pieterse (www.jkp-ads.com)
    ' Created   : 06-05-2009
    ' Purpose   : Demonstrates getting something from an OpemXML file
    '---------------------------------
    '-------------------------------------
    Dim cEditOpenXML As clsEditOpenXML
    Dim sXML As String
    Dim oXMLDoc As MSXML2.DOMDocument
    Set cEditOpenXML = New clsEditOpenXML
    With cEditOpenXML
        .CreateBackup = False
        'Tell it which OpenXML file to process
        .SourceFile = sFullFile
        'Before you can access info in the file, it must be unzipped
        .UnzipFile
        'Get XML from the ribbonX file (Office 2007 compatible)
        sXML = .GetXMLFromFile("customUI.xml", XMLFolder_customUI)
        If Len(sXML) > 0 Then
            'Change the xml of the sheet here
            Set oXMLDoc = New DOMDocument
            oXMLDoc.loadXML sXML
            oXMLDoc.Save sSaveFile
        End If
        'RibbonX for Office 2010 and up
        sXML = .GetXMLFromFile("customUI14.xml", XMLFolder_customUI)
        If Len(sXML) > 0 Then
            'Change the xml of the sheet here
            Set oXMLDoc = New DOMDocument
            oXMLDoc.loadXML sXML
            oXMLDoc.Save Replace(sSaveFile, ".xml", "14.xml")
        End If
    End With
    'Only when you let the class go out of s
    'cope the zip file's .zip extension is removed
    'in the terminate event of the class.
    'Then the OpenXML file has its original filename back.
```

```
    Set cEditOpenXML = Nothing
End Sub

Sub ClearComponent(vbComp As VBComponent)
    vbComp.CodeModule.DeleteLines 1, vbComp.CodeModule.CountOfLines
End Sub

Function WorkbookOfModule(vbComp As VBComponent) As Workbook
    Set WorkbookOfModule = WorkbookOfProject(vbComp.Collection.Parent)
End Function

Function WorkbookOfProject(vbproj As VBProject) As Workbook
    TmpStr = vbproj.FileName
    TmpStr = Right(TmpStr, Len(TmpStr) - InStrRev(TmpStr, "\"))
    Set WorkbookOfProject = Workbooks(TmpStr)
End Function

Sub DeleteComponent(vbComp As VBComponent)
    Application.DisplayAlerts = False
    If vbComp.Type = vbext_ct_Document Then
        If vbComp.Name = "ThisWorkbook" Then
            vbComp.CodeModule.DeleteLines 1, vbComp.CodeModule. _
            CountOfLines
        Else
            If WorkbookOfModule(vbComp).Sheets.Count > 1 Then
                GetSheetByCodeName(WorkbookOfModule(vbComp), vbComp.Name). _
                Delete
            Else
                Dim ws As Worksheet
                Set ws = WorkbookOfModule(vbComp).Sheets.Add
                ws.Name = "All other sheets were deleted"
                GetSheetByCodeName(WorkbookOfModule(vbComp), vbComp.Name). _
                Delete
            End If
        End If
    Else
        WorkbookOfModule(vbComp).VBProject.VBComponents.Remove vbComp
    End If
    Application.DisplayAlerts = True
End Sub

Public Sub SortListboxOnColumn(Lbox As MSForms.ListBox, OnColumn As Long)
    Dim vntData As Variant
    Dim vntTempItem As Variant
    Dim lngOuterIndex As Long
    Dim lngInnerIndex As Long
    Dim lngSubItemIndex As Long
    vntData = Lbox.List
    For lngOuterIndex = LBound(vntData, 1) To UBound(vntData, 1) - 1
        For lngInnerIndex = lngOuterIndex + 1 To UBound(vntData, 1)
            If vntData(lngOuterIndex, OnColumn) > vntData(lngInnerIndex,  _
            OnColumn) Then
```

```
                For lngSubItemIndex = 0 To Lbox.columnCount - 1
                    vntTempItem = vntData(lngOuterIndex, lngSubItemIndex)
                    vntData(lngOuterIndex, lngSubItemIndex) =  _
                    vntData(lngInnerIndex, lngSubItemIndex)
                    vntData(lngInnerIndex, lngSubItemIndex) = vntTempItem
                Next
            End If
        Next lngInnerIndex
    Next lngOuterIndex
    Lbox.Clear
    Lbox.List = vntData
End Sub

Function ComponentTypeToString(componentType As VBIDE.vbext_ComponentType) _
As String
    Select Case componentType
        Case vbext_ct_ActiveXDesigner
            ComponentTypeToString = "ActiveX Designer"
        Case vbext_ct_ClassModule
            ComponentTypeToString = "Class"
        Case vbext_ct_Document
            ComponentTypeToString = "Document"
        Case vbext_ct_MSForm
            ComponentTypeToString = "UserForm"
        Case vbext_ct_StdModule
            ComponentTypeToString = "Module"
        Case Else
            ComponentTypeToString = "Unknown Type: " & CStr(componentType)
    End Select
End Function

Public Function GetSheetByCodeName(wb As Workbook, CodeName As String) As  _
Worksheet
    Dim sh As Worksheet
    For Each sh In wb.Worksheets
        If UCase(sh.CodeName) = UCase(CodeName) Then Set  _
        GetSheetByCodeName = sh: Exit For
    Next sh
End Function

Function WorkbookIsOpen(ByVal sWbkName As String) As Boolean
    WorkbookIsOpen = False
    On Error Resume Next
    WorkbookIsOpen = Len(Workbooks(sWbkName).Name) <> 0
    On Error GoTo 0
End Function

Function GetFilePartPath(fileNameWithExtension, Optional IncludeSlash As  _
Boolean) As String
    GetFilePartPath = Left(fileNameWithExtension,  _
    InStrRev(fileNameWithExtension, "\") - 1 - IncludeSlash)
End Function
```

```
Public Function ModuleExists(Name As String, Optional ByVal _
ExistsInWorkbook As Workbook) As Boolean
    Dim j As Long
    Dim vbComp As VBComponent
    Dim modules As Collection
    Set modules = New Collection
    ModuleExists = False
    If ExistsInWorkbook Is Nothing Then
        Set ExistsInWorkbook = ThisWorkbook
    End If
    If (Name = vbNullString) Then
        GoTo errorname
    End If
    For Each vbComp In ExistsInWorkbook.VBProject.VBComponents
        If ((vbComp.Type = vbext_ct_StdModule) Or (vbComp.Type = _
        vbext_ct_ClassModule)) Then
            modules.Add vbComp.Name
        End If
    Next vbComp
    For j = 1 To modules.Count
        If (Name = modules.Item(j)) Then
            ModuleExists = True
        End If
    Next j
    j = 0
    If (ModuleExists = False) Then
        GoTo NotFound
    End If
    If (0 <> 0) Then
errorname:
        MsgBox ("Function BootStrap.Is_Module_Loaded Was not passed a _
        Name of Module")
        Exit Function
    End If
    If (0 <> 0) Then
NotFound:
        Exit Function
    End If
End Function

Function GetFilePartName(fileNameWithExtension As String, Optional _
IncludeExtension As Boolean) As String
    If InStr(1, fileNameWithExtension, "\") > 0 Then
        GetFilePartName = Right(fileNameWithExtension, _
        Len(fileNameWithExtension) - InStrRev(fileNameWithExtension, "\"))
    Else
        GetFilePartName = fileNameWithExtension
    End If
    If IncludeExtension = False Then GetFilePartName = _
    Left(GetFilePartName, InStr(1, GetFilePartName, ".") - 1)
End Function
```

```
Public Function IsArrayAllocated(ByRef arr As Variant) As Boolean
    On Error Resume Next
    IsArrayAllocated = IsArray(arr) And (Not IsError(LBound(arr, 1))) And  _
    LBound(arr, 1) <= UBound(arr, 1)
End Function

Public Function GetFilePath(Optional FileType As Variant, Optional  _
multiSelect As Boolean) As Variant
    Dim blArray As Boolean
    Dim i As Long
    Dim strErrMsg As String, strTitle As String
    Dim varItem As Variant
    If Not IsMissing(FileType) Then
        blArray = IsArray(FileType)
        If Not blArray Then strErrMsg = "Please pass an array in the  _
        first parameter of this function!"
        If IsArrayAllocated(FileType) = False Then blArray = False
    End If
    If strErrMsg = vbNullString Then
        If multiSelect Then strTitle = "Choose one or more files" Else  _
        strTitle = "Choose file"
        With Application.FileDialog(msoFileDialogFilePicker)
            .InitialFileName = Environ("USERprofile") & "\Desktop\"
            .AllowMultiSelect = multiSelect
            .Filters.Clear
            If blArray Then .Filters.Add "File type", Replace("*." &  _
            Join(FileType, ", *."), "..", ".")
            .Title = strTitle
            If .Show <> 0 Then
                ReDim arrResults(1 To .SelectedItems.Count) As Variant
                For Each varItem In .SelectedItems
                    i = i + 1
                    arrResults(i) = varItem
                Next varItem
                GetFilePath = arrResults
            End If
        End With
    Else
        MsgBox strErrMsg, vbCritical, "Error!"
    End If
End Function

Rem Listbox
Public Function ListboxContains(Lbox As MSForms.ListBox, str As String, _
Optional ColumnIndexZeroBased As Long = -1, _
Optional CaseSensitive As Boolean = False) As Boolean
    Dim i      As Long
    Dim n      As Long
    Dim sTemp  As String
    If ColumnIndexZeroBased > Lbox.columnCount - 1 Or  _
    ColumnIndexZeroBased < 0 Then
```

```
            ColumnIndexZeroBased = -1
        End If
        n = Lbox.ListCount
        If ColumnIndexZeroBased <> -1 Then
            For i = n - 1 To 0 Step -1
                If CaseSensitive = True Then
                    sTemp = Lbox.List(i, ColumnIndexZeroBased)
                Else
                    str = LCase(str)
                    sTemp = LCase(Lbox.List(i, ColumnIndexZeroBased))
                End If
                If InStr(1, sTemp, str) > 0 Then
                    ListboxContains = True
                    Exit Function
                End If
            Next i
        Else
            Dim columnCount As Long
            n = Lbox.ListCount
            For i = n - 1 To 0 Step -1
                For columnCount = 0 To Lbox.columnCount - 1
                    If CaseSensitive = True Then
                        sTemp = Lbox.List(i, columnCount)
                    Else
                        str = LCase(str)
                        sTemp = LCase(Lbox.List(i, columnCount))
                    End If
                    If InStr(1, sTemp, str) > 0 Then
                        ListboxContains = True
                        Exit Function
                    End If
                Next columnCount
            Next i
        End If
    End Function

    Function ProtectedVBProject(ByVal wb As Workbook) As Boolean
        If wb.VBProject.Protection = 1 Then
            ProtectedVBProject = True
        Else
            ProtectedVBProject = False
        End If
    End Function

    Function SheetAdd(SheetName As String, TargetWorkbook As Workbook) As _
    Worksheet
        Dim NewSheet As Worksheet
        If WorksheetExists(SheetName, TargetWorkbook) = True Then
            Set SheetAdd = TargetWorkbook.Sheets(SheetName)
        Else
            Set SheetAdd = TargetWorkbook.Sheets.Add
            SheetAdd.Name = SheetName
```

```
    End If
End Function

Function WorksheetExists(shtName As String, Optional wb As Workbook) As _
Boolean
    Dim sht As Worksheet
    If wb Is Nothing Then
        Set wb = ThisWorkbook
    End If
    On Error Resume Next
    Set sht = wb.Sheets(shtName)
    On Error GoTo 0
    WorksheetExists = Not sht Is Nothing
End Function

Sub ResizeUserformToFitControls(Form As Object)
    Dim ctr As MSForms.control
    Dim myWidth
    myWidth = Form.InsideWidth
    For Each ctr In Form.Controls
        If ctr.Left + ctr.Width > myWidth Then myWidth = ctr.Left + ctr. _
        Width
    Next
    Form.Width = myWidth + Form.Width - Form.InsideWidth        '+ 10
    Dim myHeight
    myHeight = Form.InsideHeight
    For Each ctr In Form.Controls
        If ctr.Top + ctr.Height > myHeight Then myHeight = ctr.Top + ctr. _
        Height
    Next
    Form.Height = myHeight + Form.Height - Form.InsideHeight        '+ 10
End Sub

Sub ResizeControlColumns(ctr As MSForms.control, Optional ResizeListbox _
As Boolean)
    If ctr.ListCount = 0 Then Exit Sub
    Application.ScreenUpdating = False
    Dim ws As Worksheet
    Set ws = SheetAdd("ListboxColumnwidth", ThisWorkbook)
    '---Listbox to range-----
    Dim rng As Range
    Set rng = ThisWorkbook.Sheets("ListboxColumnwidth").Range("A1")
    Set rng = rng.Resize(UBound(ctr.List) + 1, ctr.columnCount)
    rng = ctr.List

    '---Get ColumnWidths------
    rng.Columns.AutoFit
    Dim sWidth As String
    Dim vR() As Variant
    Dim n As Integer
    Dim cell As Range
    For Each cell In rng.Resize(1)
```

```
            n = n + 1
            ReDim Preserve vR(1 To n)
            vR(n) = cell.EntireColumn.Width
        Next cell
        sWidth = Join(vR, ";")
        'Debug.Print sWidth

        '---assign ColumnWidths----
        With ctr
            .ColumnWidths = sWidth
            '.RowSource = "A1:A3"
            .BorderStyle = fmBorderStyleSingle
        End With

        'remove worksheet
        Application.DisplayAlerts = False
        ws.Delete
        Application.DisplayAlerts = True

        Application.ScreenUpdating = True

        '----Resize Listbox--------
        If ResizeListbox = False Then Exit Sub
        Dim w As Long
        For i = LBound(vR) To UBound(vR)
            w = w + vR(i)
        Next
        DoEvents
        ctr.Width = w + 10
    End Sub

    Sub FoldersCreate(FolderPath As String)
        Dim individualFolders() As String
        Dim tempFolderPath As String
        Dim arrayElement As Variant
        individualFolders = Split(FolderPath, "\")
        For Each arrayElement In individualFolders
            tempFolderPath = tempFolderPath & arrayElement & "\"
            If FolderExists(tempFolderPath) = False Then
                MkDir tempFolderPath
            End If
        Next arrayElement
    End Sub

    Function FolderExists(ByVal strPath As String) As Boolean
        On Error Resume Next
        FolderExists = ((GetAttr(strPath) And vbDirectory) = vbDirectory)
        On Error GoTo 0
    End Function

    Sub FollowLink(FolderPath As String)
        Dim oShell As Object
```

```
    Dim Wnd As Object
    Set oShell = CreateObject("Shell.Application")
    For Each Wnd In oShell.Windows
        If Wnd.Name = "File Explorer" Then
            If Wnd.Document.Folder.Self.Path = FolderPath Then Exit Sub
        End If
    Next Wnd
    Application.ThisWorkbook.FollowHyperlink Address:=FolderPath,  _
    NewWindow:=True
End Sub

Function InStrExact(Start As Long, SourceText As String, WordToFind As  _
String, _
Optional CaseSensitive As Boolean = False, _
Optional AllowAccentedCharacters As Boolean = False) As Long
    Dim X As Long, Str1 As String, Str2 As String, Pattern As String
    Const UpperAccentsOnly As String = "HIP"
    Const UpperAndLowerAccents As String = "HIPηιρ"
    If CaseSensitive Then
        Str1 = SourceText
        Str2 = WordToFind
        Pattern = "[!A-Za-z0-9]"
        If AllowAccentedCharacters Then Pattern = Replace(Pattern, "!",  _
        "!" & UpperAndLowerAccents)
    Else
        Str1 = UCase(SourceText)
        Str2 = UCase(WordToFind)
        Pattern = "[!A-Z0-9]"
        If AllowAccentedCharacters Then Pattern = Replace(Pattern, "!",  _
        "!" & UpperAccentsOnly)
    End If
    For X = Start To Len(Str1) - Len(Str2) + 1
        If Mid(" " & Str1 & " ", X, Len(Str2) + 2) Like Pattern & Str2 &  _
        Pattern _
        And Not Mid(Str1, X) Like Str2 & "'[" & Mid(Pattern, 3) & "*" Then
            InStrExact = X
            Exit Function
        End If
    Next
End Function

Function TxtAppend(sFile As String, sText As String)
    On Error GoTo Err_Handler
    Dim iFileNumber           As Integer
    iFileNumber = FreeFile
    Open sFile For Append As #iFileNumber
    Print #iFileNumber, sText
    Close #iFileNumber
Exit_Err_Handler:
    Exit Function
Err_Handler:
    MsgBox "The following error has occurred" & vbCrLf & vbCrLf & _
```

```
        "Error Number: " & err.Number & vbCrLf & _
        "Error Source: Txt_Append" & vbCrLf & _
        "Error Description: " & err.Description & _
        Switch(Erl = 0, "", Erl <> 0, vbCrLf & "Line No: " & Erl) _
        , vbOKOnly + vbCritical, "An Error has Occurred!"
        GoTo Exit_Err_Handler
    End Function

    Function GetCompText(vbComp As VBComponent) As String
        Dim codeMod As CodeModule
        Set codeMod = vbComp.CodeModule
        If codeMod.CountOfLines = 0 Then GetCompText = "": Exit Function
        GetCompText = codeMod.Lines(1, codeMod.CountOfLines)
    End Function

    Function getDeclarations(wb As Workbook) As Collection
        Dim Output As Collection: Set Output = New Collection
        Dim declarationsCollection As Collection: Set declarationsCollection  _
        = New Collection
        Dim keywordsCollection As Collection: Set keywordsCollection = New  _
        Collection
        Dim vbComp As VBComponent
        Dim codeMod As CodeModule
        Dim str As Variant
        Dim i As Long
        Dim element As Variant
        Dim originalDeclarations As Variant
        Dim tmp As String
        Dim helper As String
        Dim typeCollection As Collection: Set typeCollection = New Collection
        Dim componentCollection As Collection: Set componentCollection = New  _
        Collection
        Dim componentTypeCollection As Collection: Set  _
        componentTypeCollection = New Collection
        Dim scopeCollection As Collection: Set scopeCollection = New  _
        Collection
        For Each vbComp In wb.VBProject.VBComponents
            'If vbComp.Type <> vbext_ct_ClassModule Then
            Set codeMod = vbComp.CodeModule
            If codeMod.CountOfDeclarationLines > 0 Then
                str = codeMod.Lines(1, codeMod.CountOfDeclarationLines)
                str = Replace(str, "_" & vbNewLine, "")
                originalDeclarations = str
                tmp = str
                Do While InStr(1, str, "End Type") > 0
                    tmp = Mid(str, InStr(1, str, "Type "), InStr(1, str, "End  _
                    Type") - InStr(1, str, "Type ") + 8)
                    str = Replace(str, tmp, Split(tmp, vbNewLine)(0))
                Loop
                Do While InStr(1, str, "End Enum") > 0
                    tmp = Mid(str, InStr(1, str, "Enum "), InStr(1, str, "End  _
                    Enum") - InStr(1, str, "Enum ") + 8)
```

```
            str = Replace(str, tmp, Split(tmp, vbNewLine)(0))
        Loop
        Do While InStr(1, str, "  ") > 0
            str = Replace(str, "  ", " ")
        Loop
        str = Split(str, vbNewLine)
        tmp = originalDeclarations
        For Each element In str
            If InStr(1, CStr(element), "Enum ", vbTextCompare) > 0  _
            Then
                keywordsCollection.Add getWord(CStr(element), " ",  _
                "Enum")
                declarationsCollection.Add getWord(tmp, , "Enum " &  _
                keywordsCollection.Item(keywordsCollection.Count),  _
                "End Enum", , , True)
                typeCollection.Add "Enum"
                componentCollection.Add vbComp.Name
                componentTypeCollection.Add  _
                ComponentTypeToString(vbComp.Type)
                scopeCollection.Add IIf(InStr(1,  _
                declarationsCollection.Item(declarationsCollection. _
                Count), "Public", vbTextCompare), "Public", "Private")
            ElseIf InStr(1, CStr(element), "Type ", vbTextCompare) >  _
            0 Then
                keywordsCollection.Add getWord(CStr(element), " ",  _
                "Type")
                declarationsCollection.Add getWord(tmp, , "Type " &  _
                keywordsCollection.Item(keywordsCollection.Count),  _
                "End Type", , , True)
                typeCollection.Add "Type"
                componentCollection.Add vbComp.Name
                componentTypeCollection.Add  _
                ComponentTypeToString(vbComp.Type)
                scopeCollection.Add IIf(InStr(1,  _
                declarationsCollection.Item(declarationsCollection. _
                Count), "Public", vbTextCompare), "Public", "Private")
            ElseIf InStr(1, CStr(element), "Const ", vbTextCompare) >  _
            0 Then
                keywordsCollection.Add getWord(CStr(element), " ",  _
                "Const")
                declarationsCollection.Add CStr(element)
                typeCollection.Add "Const"
                componentCollection.Add vbComp.Name
                componentTypeCollection.Add  _
                ComponentTypeToString(vbComp.Type)
                scopeCollection.Add IIf(InStr(1,  _
                declarationsCollection.Item(declarationsCollection. _
                Count), "Public", vbTextCompare), "Public", "Private")
            ElseIf InStr(1, CStr(element), "Sub ", vbTextCompare) > 0  _
            Then
                keywordsCollection.Add getWord(CStr(element), " ",  _
                "Sub")
```

```
                declarationsCollection.Add CStr(element)
                typeCollection.Add "Sub"
                componentCollection.Add vbComp.Name
                componentTypeCollection.Add  _
                ComponentTypeToString(vbComp.Type)
                scopeCollection.Add IIf(InStr(1,  _
                declarationsCollection.Item(declarationsCollection. _
                Count), "Public", vbTextCompare), "Public", "Private")
            ElseIf InStr(1, CStr(element), "Function ", vbTextCompare) _
            > 0 Then
                keywordsCollection.Add getWord(CStr(element), " ",  _
                "Function")
                declarationsCollection.Add CStr(element)
                typeCollection.Add "Function"
                componentCollection.Add vbComp.Name
                componentTypeCollection.Add  _
                ComponentTypeToString(vbComp.Type)
                scopeCollection.Add IIf(InStr(1,  _
                declarationsCollection.Item(declarationsCollection. _
                Count), "Public", vbTextCompare), "Public", "Private")
            ElseIf element Like "*(*) As *" Then
                helper = Left(element, InStr(1, CStr(element), "(") -  _
                1)
                helper = Mid(helper, InStrRev(helper, " ") + 1)
                keywordsCollection.Add helper
                declarationsCollection.Add CStr(element)
                typeCollection.Add "Other"
                componentCollection.Add vbComp.Name
                componentTypeCollection.Add  _
                ComponentTypeToString(vbComp.Type)
                scopeCollection.Add IIf(InStr(1,  _
                declarationsCollection.Item(declarationsCollection. _
                Count), "Public", vbTextCompare), "Public", "Private")
            ElseIf element Like "* As *" Then
                keywordsCollection.Add getWord(CStr(element), " ", ,  _
                "As")
                declarationsCollection.Add CStr(element)
                typeCollection.Add "Other"
                componentCollection.Add vbComp.Name
                componentTypeCollection.Add  _
                ComponentTypeToString(vbComp.Type)
                scopeCollection.Add IIf(InStr(1,  _
                declarationsCollection.Item(declarationsCollection. _
                Count), "Public", vbTextCompare), "Public", "Private")
            Else
                'nothing
            End If
        Next element
    End If
     'End If
Next vbComp
Output.Add declarationsCollection
```

```
    Output.Add scopeCollection
    Output.Add typeCollection
    Output.Add keywordsCollection
    Output.Add componentTypeCollection
    Output.Add componentCollection
    Set getDeclarations = Output
End Function

Function getWord(str As Variant, Optional delim As String _
, Optional afterWord As String _
, Optional beforeWord As String _
, Optional counter As Integer _
, Optional outer As Boolean _
, Optional includeWords As Boolean) As String
    Dim i As Long
    If afterWord = "" And beforeWord = "" And counter = 0 Then MsgBox  _
    ("Pass at least 1 parameter betweenn -AfterWord- , -BeforeWord- , - _
    counter-"): Exit Function
    If TypeName(str) = "String" Then
        If delim <> "" Then
            str = Split(str, delim)
        If UBound(str) <> 0 Then
            If afterWord = "" And beforeWord = "" And counter <> 0  _
            Then If counter - 1 <= UBound(str) Then getWord =  _
            str(counter - 1): Exit Function
            For i = LBound(str) To UBound(str)
                If afterWord <> "" And beforeWord = "" Then If i <> 0  _
                Then If str(i - 1) = afterWord Then getWord = str(i):  _
                Exit Function
                If afterWord = "" And beforeWord <> "" Then If i <>  _
                UBound(str) Then If str(i + 1) = beforeWord Then  _
                getWord = str(i): Exit Function
                If afterWord <> "" And beforeWord <> "" Then If i <>  _
                0 And i <> UBound(str) Then If str(i - 1) = afterWord  _
                And str(i + 1) = beforeWord Then getWord = str(i):  _
                Exit Function
            Next i
        End If
        Else
            'If afterWord <> "" And beforeWord <> "" then
            If InStr(1, str, afterWord) > 0 And InStr(1, str, beforeWord)  _
            > 0 Then
                If includeWords = False Then
                    getWord = Mid(str, InStr(1, str, afterWord) +  _
                    Len(afterWord))
                Else
                    getWord = Mid(str, InStr(1, str, afterWord))
                End If
                If outer = True Then
                    If includeWords = False Then
                        getWord = Left(getWord, InStrRev(getWord,  _
                        beforeWord) - 1)
```

```
                    Else
                        getWord = Left(getWord, InStrRev(getWord,  _
                        beforeWord) + Len(beforeWord) - 1)
                    End If
                Else
                    If includeWords = False Then
                        getWord = Left(getWord, InStr(1, getWord,  _
                        beforeWord) - 1)
                    Else
                        getWord = Left(getWord, InStr(1, getWord,  _
                        beforeWord) + Len(beforeWord) - 1)
                    End If
                End If
                Exit Function
            End If
            'ElseIf afterWord <> "" And beforeWord = "" then
            'getWord = Mid(str, InStr(1, str, afterWord))
            'elseIf afterWord = "" And beforeWord <> "" then
        End If
    Else
    End If
    getWord = vbNullString
End Function

Function CollectionsToArrayTable(collections As Collection) As Variant
    Dim columnCount As Long
    columnCount = collections.Count
    Dim rowCount As Long
    rowCount = collections.Item(1).Count
    Dim var As Variant
    ReDim var(1 To rowCount, 1 To columnCount)
    Dim cols As Long
    Dim rows As Long
    For rows = 1 To rowCount
        For cols = 1 To collections.Count
            var(rows, cols) = collections(cols).Item(rows)
        Next cols
    Next rows
    CollectionsToArrayTable = var
End Function

Public Function ArrayDimensionLength(SourceArray As Variant) As Integer
    Dim i As Integer
    Dim test As Long
    On Error GoTo catch
    Do
        i = i + 1
        test = UBound(SourceArray, i)
    Loop
catch:
    ArrayDimensionLength = i - 1
End Function
```

```
Rem @AUTHOR ROBERT TODAR
Public Function ArrayToString(SourceArray As Variant, Optional Delimiter _
As String = ",") As String
    Dim temp As String
    Select Case ArrayDimensionLength(SourceArray)
        Case 1
            temp = Join(SourceArray, Delimiter)
        Case 2
            Dim RowIndex As Long
            Dim ColIndex As Long
            For RowIndex = LBound(SourceArray, 1) To UBound(SourceArray, _
            1)
                For ColIndex = LBound(SourceArray, 2) To _
                UBound(SourceArray, 2)
                    temp = temp & SourceArray(RowIndex, ColIndex)
                    If ColIndex <> UBound(SourceArray, 2) Then temp = _
                    temp & Delimiter
                Next ColIndex
                If RowIndex <> UBound(SourceArray, 1) Then temp = temp & _
                vbNewLine
            Next RowIndex
    End Select
    ArrayToString = temp
End Function

Function ProcListCollection(Module As VBComponent) As Collection
    Dim coll As Collection: Set coll = New Collection
    Dim lineNum As Long, NumLines As Long
    Dim ProcName As String
    Dim ProcKind As VBIDE.vbext_ProcKind
    lineNum = Module.CodeModule.CountOfDeclarationLines + 1
    Do Until lineNum >= Module.CodeModule.CountOfLines
        ProcName = Module.CodeModule.ProcOfLine(lineNum, ProcKind)
        coll.Add ProcName
        lineNum = Module.CodeModule.ProcStartLine(ProcName, ProcKind) + _
        Module.CodeModule.ProcCountLines(ProcName, ProcKind) + 1
    Loop
    Set ProcListCollection = coll
End Function

Public Function GetProcText(vbComp As VBComponent, _
sProcName As String, _
Optional bInclHeader As Boolean = True)
    Dim codeMod As CodeModule
    Set codeMod = vbComp.CodeModule

    Dim lProcStart          As Long
    Dim lProcBodyStart      As Long
    Dim lProcNoLines        As Long
    Const vbext_pk_Proc = 0
    On Error GoTo error_handler
```

```
        lProcStart = codeMod.ProcStartLine(sProcName, vbext_pk_Proc)
        lProcBodyStart = codeMod.ProcBodyLine(sProcName, vbext_pk_Proc)
        lProcNoLines = codeMod.ProcCountLines(sProcName, vbext_pk_Proc)
        If bInclHeader = True Then
            GetProcText = codeMod.Lines(lProcStart, lProcNoLines)
        Else
            lProcNoLines = lProcNoLines - (lProcBodyStart - lProcStart)
            GetProcText = codeMod.Lines(lProcBodyStart, lProcNoLines)
        End If
    Error_Handler_Exit:
        On Error Resume Next
        Exit Function
    error_handler:

        'Debug.Print "The following error has occurred" & vbCrLf & vbCrLf & _
        "Error Number: " & err.Number & vbCrLf & _
        "Error Source: GetProcText" & vbCrLf & _
        "Error Description: " & err.Description & _
        Switch(Erl = 0, vbNullString, Erl <> 0, vbCrLf & "Line No: " & Erl)
        Resume Error_Handler_Exit
    End Function

    Rem @todo
    Public Function IndentModule(Optional vbComp As VBComponent)
        If vbComp Is Nothing Then Set vbComp = ActiveModule
        If vbComp.CodeModule.CountOfLines = 0 Then Exit Function
        Dim nIndent As Integer
        Dim nLine As Long
        Dim strNewLine As String
        For nLine = 1 To vbComp.CodeModule.CountOfLines
            strNewLine = vbComp.CodeModule.Lines(nLine, 1)
            strNewLine = LTrim$(strNewLine)
            If IsBlockEnd(strNewLine) Then nIndent = nIndent - 1
            If nIndent < 0 Then nIndent = 0
            If strNewLine <> "" Then vbComp.CodeModule.ReplaceLine nLine,  _
            Space$(nIndent * 4) & strNewLine
            If IsBlockStart(strNewLine) Then nIndent = nIndent + 1
        Next nLine
    End Function

    Public Function IndentProcedure()
        Dim Module As VBComponent: Set Module = ActiveModule
        If Module.CodeModule.CountOfLines = 0 Then Exit Function
        Dim ProcedureName As String: ProcedureName = ActiveProcName
        Dim FirstLine As Long: FirstLine = Module.CodeModule. _
        ProcStartLine(ProcedureName, vbext_pk_Proc)
        Dim EndLine As Long: EndLine = ProcedureEndLine(Module, ProcedureName)
        Rem = Module.CodeModule.ProcCountLines(ProcedureName, vbext_pk_Proc)
        Rem = ProcedureType(WorkbookOfModule(Module), ProcedureName)
        Dim nIndent As Integer
        Dim nLine As Long
        Dim strNewLine As String
```

```
    For nLine = FirstLine To EndLine
        strNewLine = Module.CodeModule.Lines(nLine, 1)
        strNewLine = LTrim$(strNewLine)
        If IsBlockEnd(strNewLine) Then nIndent = nIndent - 1
        If nIndent < 0 Then nIndent = 0
        If strNewLine <> "" Then Module.CodeModule.ReplaceLine nLine,  _
        Space$(nIndent * 4) & strNewLine
        If IsBlockStart(strNewLine) Then nIndent = nIndent + 1
    Next nLine
End Function

Public Function IndentWorkbook(Optional wb As Workbook)
    If wb Is Nothing Then Set wb = ActiveCodepaneWorkbook
    Dim vbComp As VBComponent
    For Each vbComp In wb.VBProject.VBComponents
        IndentModule vbComp
    Next
End Function

Public Function IsBlockEnd(strLine As String) As Boolean
    Dim bOK As Boolean
    Dim nPos As Integer
    Dim strTemp As String
    nPos = InStr(1, strLine, " ") - 1
    If nPos < 0 Then nPos = Len(strLine)
    strTemp = Left$(strLine, nPos)
    Select Case strTemp
        Case "Next", "Loop", "Wend", "End Select", "Case", "Else",  _
        "#Else", "Else:", "#Else:", "ElseIf", "#ElseIf", "End If", "#End  _
        If"
            bOK = True
        Case "End"
            bOK = (Len(strLine) > 3)
    End Select
    IsBlockEnd = bOK
End Function

Public Function IsBlockStart(strLine As String) As Boolean
    Dim bOK As Boolean
    Dim nPos As Integer
    Dim strTemp As String
    nPos = InStr(1, strLine, " ") - 1
    If nPos < 0 Then nPos = Len(strLine)
    strTemp = Left$(strLine, nPos)
    Select Case strTemp
        Case "With", "For", "Do", "While", "Select", "Case", "Else",  _
        "Else:", "#Else", "#Else:", "Sub", "Function", "Property", "Enum", _
        "Type"
            bOK = True
        Case "If", "#If", "ElseIf", "#ElseIf"
            bOK = (Len(strLine) = (InStr(1, strLine, " Then") + 4))
        Case "public", "Public", "Friend"
```

```
            nPos = InStr(1, strLine, " Static ")
            If nPos Then
                nPos = InStr(nPos + 7, strLine, " ")
            Else
                nPos = InStr(Len(strTemp) + 1, strLine, " ")
            End If
            Select Case Mid$(strLine, nPos + 1, InStr(nPos + 1, strLine,  _
            " ") - nPos - 1)
                Case "Sub", "Function", "Property", "Enum", "Type"
                    bOK = True
            End Select
        End Select
        IsBlockStart = bOK
    End Function

    Function CollectionToArray(C As Collection) As Variant
        Dim A() As Variant: ReDim A(0 To C.Count - 1)
        Dim i As Long
        For i = 1 To C.Count
            A(i - 1) = C.Item(i)
        Next
        CollectionToArray = A
    End Function
```

--- mFormProperties ---

```
'''''''''''''''''''
Rem JKP
' Company    : JKP Application Development Services (c)
' Author     : Jan Karel Pieterse (www.jkp-ads.com)
Sub ListFormsExport(oWb As Workbook, sExportPath As String)
    Dim oVBProj As VBProject
    Dim cControl As control
    Dim oComp As VBComponent
    Dim oSh As Worksheet
    Dim lCount As Long
    Application.Calculation = xlCalculationManual
    Application.ScreenUpdating = False
    On Error GoTo LocErr
    Set oVBProj = oWb.VBProject
    Dim tmpBook As Workbook
    Set tmpBook = Workbooks.Add
    ThisWorkbook.Worksheets("Userforms").Copy Before:=tmpBook.Sheets(1)
    Set oSh = tmpBook.Sheets("Userforms")
    oSh.Visible = xlSheetVisible
    Dim sh As Worksheet
    Application.DisplayAlerts = False
    For Each sh In tmpBook.Sheets
        If sh.Name <> "Userforms" Then sh.Delete
    Next
    Application.DisplayAlerts = True
    For Each oComp In oVBProj.VBComponents
        If oComp.Name <> "uProjectManager" Then
            If oComp.Type = vbext_ct_MSForm Then
                Application.StatusBar = oComp.Name
                ListProperties oComp.Name, oComp.Designer, oSh, lCount
                lCount = lCount + 1
                For Each cControl In oComp.Designer.Controls
                    ListProperties oComp.Name, cControl, oSh, lCount
                    lCount = lCount + 1
                Next
            End If
        End If
    Next
    If Right(sExportPath, 1) <> Application.PathSeparator Then  _
    sExportPath = sExportPath & Application.PathSeparator
    Application.DisplayAlerts = False
    '    tmpBook.SaveAs sExportPath &
    '"FormsAndControlsProperties.txt", xlCSV
    '    tmpBook.Close False
    Application.StatusBar = False
    Application.Calculation = xlCalculationAutomatic
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    Exit Sub
LocErr:
    Debug.Print err.Description
```

```
    Stop
    Resume        'next
End Sub

Sub ListProperties(ByVal sFormName As String, ByVal cControl As Object,  _
oSh As Worksheet, lCount As Long)
    On Error Resume Next
    With oSh
        .Names("Userforms.Form").RefersToRange.Offset(lCount, 0) =  _
        sFormName
        .Names("Userforms.Controlname").RefersToRange.Offset(lCount, 0) =  _
        cControl.Name
        .Names("Userforms.NewName").RefersToRange.Offset(lCount, 0) =  _
        cControl.Name        'Remember name
        .Names("Userforms.Type").RefersToRange.Offset(lCount, 0) =  _
        TypeName(cControl)
        .Names("Userforms.Accelerator").RefersToRange.Offset(lCount, 0) =  _
        cControl.Accelerator
        .Names("Userforms.ActiveControl").RefersToRange.Offset(lCount, 0)  _
        = cControl.ActiveControl
        .Names("Userforms.Alignment").RefersToRange.Offset(lCount, 0) =  _
        cControl.Alignment
        .Names("Userforms.AutoSize").RefersToRange.Offset(lCount, 0) =  _
        cControl.AutoSize
        .Names("Userforms.BackColor").RefersToRange.Offset(lCount, 0) =  _
        cControl.BackColor
        .Names("Userforms.BackStyle").RefersToRange.Offset(lCount, 0) =  _
        cControl.BackStyle
        .Names("Userforms.BorderColor").RefersToRange.Offset(lCount, 0) =  _
        cControl.BorderColor
        .Names("Userforms.BorderStyle").RefersToRange.Offset(lCount, 0) =  _
        cControl.BorderStyle
        .Names("Userforms.BoundColumn").RefersToRange.Offset(lCount, 0) =  _
        cControl.BoundColumn
        .Names("Userforms.BoundValue").RefersToRange.Offset(lCount, 0) =  _
        cControl.BoundValue
        .Names("Userforms.Cancel").RefersToRange.Offset(lCount, 0) =  _
        cControl.Cancel
        .Names("Userforms.CanPaste").RefersToRange.Offset(lCount, 0) =  _
        cControl.CanPaste
        .Names("Userforms.CanRedo").RefersToRange.Offset(lCount, 0) =  _
        cControl.CanRedo
        .Names("Userforms.CanUndo").RefersToRange.Offset(lCount, 0) =  _
        cControl.CanUndo
        .Names("Userforms.Caption").RefersToRange.Offset(lCount, 0) =  _
        cControl.Caption
        .Names("Userforms.Column").RefersToRange.Offset(lCount, 0) =  _
        cControl.Column
        .Names("Userforms.ColumnCount").RefersToRange.Offset(lCount, 0) =  _
        cControl.columnCount
        .Names("Userforms.ColumnHeads").RefersToRange.Offset(lCount, 0) =  _
        cControl.ColumnHeads
```

```
        .Names("Userforms.ColumnWidths").RefersToRange.Offset(lCount, 0)  _
        = cControl.ColumnWidths
        .Names("Userforms.ControlSource").RefersToRange.Offset(lCount, 0)  _
        = cControl.ControlSource
        .Names("Userforms.ControlTipText").RefersToRange.Offset(lCount, 0) _
        = cControl.ControlTipText
        .Names("Userforms.Cycle").RefersToRange.Offset(lCount, 0) =  _
        cControl.Cycle
        .Names("Userforms.Default").RefersToRange.Offset(lCount, 0) =  _
        cControl.Default
        .Names("Userforms.DrawBuffer").RefersToRange.Offset(lCount, 0) =  _
        cControl.DrawBuffer
        .Names("Userforms.Enabled").RefersToRange.Offset(lCount, 0) =  _
        cControl.Enabled
        .Names("Userforms.FontName").RefersToRange.Offset(lCount, 0) =  _
        cControl.Font.Name
        .Names("Userforms.FontSize").RefersToRange.Offset(lCount, 0) =  _
        cControl.Font.size
        .Names("Userforms.ForeColor").RefersToRange.Offset(lCount, 0) =  _
        cControl.ForeColor
        .Names("Userforms.GroupName").RefersToRange.Offset(lCount, 0) =  _
        cControl.GroupName
        .Names("Userforms.Height").RefersToRange.Offset(lCount, 0) =  _
        cControl.Height
        .Names("Userforms.HelpContextID").RefersToRange.Offset(lCount, 0)  _
        = cControl.HelpContextID
        .Names("Userforms.IMEMode").RefersToRange.Offset(lCount, 0) =  _
        cControl.IMEMode
        .Names("Userforms.InsideHeight").RefersToRange.Offset(lCount, 0)  _
        = cControl.InsideHeight
        .Names("Userforms.InsideWidth").RefersToRange.Offset(lCount, 0) =  _
        cControl.InsideWidth
        .Names("Userforms.IntegralHeight").RefersToRange.Offset(lCount, 0) _
        = cControl.IntegralHeight
        .Names("Userforms.KeepScrollBarsVisible").RefersToRange. _
        Offset(lCount, 0) = cControl.KeepScrollBarsVisible
        .Names("Userforms.LayoutEffect").RefersToRange.Offset(lCount, 0)  _
        = cControl.LayoutEffect
        .Names("Userforms.Left").RefersToRange.Offset(lCount, 0) =  _
        cControl.Left
        .Names("Userforms.List").RefersToRange.Offset(lCount, 0) =  _
        cControl.List
        .Names("Userforms.ListCount").RefersToRange.Offset(lCount, 0) =  _
        cControl.ListCount
        .Names("Userforms.ListIndex").RefersToRange.Offset(lCount, 0) =  _
        cControl.ListIndex
        .Names("Userforms.ListStyle").RefersToRange.Offset(lCount, 0) =  _
        cControl.ListStyle
        .Names("Userforms.Locked").RefersToRange.Offset(lCount, 0) =  _
        cControl.Locked
        .Names("Userforms.MatchEntry").RefersToRange.Offset(lCount, 0) =  _
        cControl.MatchEntry
```

```
        .Names("Userforms.MouseIcon").RefersToRange.Offset(lCount, 0) =  _
        cControl.MouseIcon
        .Names("Userforms.MousePointer").RefersToRange.Offset(lCount, 0)  _
        = cControl.MousePointer
        .Names("Userforms.MultiSelect").RefersToRange.Offset(lCount, 0) =  _
        cControl.multiSelect
        .Names("Userforms.Object").RefersToRange.Offset(lCount, 0) =  _
        cControl.Object
        .Names("Userforms.OldHeight").RefersToRange.Offset(lCount, 0) =  _
        cControl.OldHeight
        .Names("Userforms.OldLeft").RefersToRange.Offset(lCount, 0) =  _
        cControl.OldLeft
        .Names("Userforms.OldWidth").RefersToRange.Offset(lCount, 0) =  _
        cControl.OldWidth
        .Names("Userforms.Parent").RefersToRange.Offset(lCount, 0) =  _
        cControl.Parent
        .Names("Userforms.Picture").RefersToRange.Offset(lCount, 0) =  _
        cControl.Picture
        .Names("Userforms.PictureAlignment").RefersToRange.Offset(lCount,  _
        0) = cControl.PictureAlignment
        .Names("Userforms.PicturePosition").RefersToRange.Offset(lCount,  _
        0) = cControl.PicturePosition
        .Names("Userforms.PictureSizeMode").RefersToRange.Offset(lCount,  _
        0) = cControl.PictureSizeMode
        .Names("Userforms.PictureTiling").RefersToRange.Offset(lCount, 0)  _
        = cControl.PictureTiling
        .Names("Userforms.RowSource").RefersToRange.Offset(lCount, 0) =  _
        cControl.RowSource
        .Names("Userforms.ScrollBars").RefersToRange.Offset(lCount, 0) =  _
        cControl.ScrollBars
        .Names("Userforms.ScrollHeight").RefersToRange.Offset(lCount, 0)  _
        = cControl.ScrollHeight
        .Names("Userforms.ScrollLeft").RefersToRange.Offset(lCount, 0) =  _
        cControl.ScrollLeft
        .Names("Userforms.ScrollTop").RefersToRange.Offset(lCount, 0) =  _
        cControl.ScrollTop
        .Names("Userforms.ScrollWidth").RefersToRange.Offset(lCount, 0) =  _
        cControl.ScrollWidth
        .Names("Userforms.Selected").RefersToRange.Offset(lCount, 0) =  _
        cControl.Selected
        .Names("Userforms.SpecialEffect").RefersToRange.Offset(lCount, 0)  _
        = cControl.SpecialEffect
        .Names("Userforms.TabIndex").RefersToRange.Offset(lCount, 0) =  _
        cControl.TabIndex
        .Names("Userforms.TabStop").RefersToRange.Offset(lCount, 0) =  _
        cControl.TabStop
        .Names("Userforms.Tag").RefersToRange.Offset(lCount, 0) =  _
        cControl.Tag
        .Names("Userforms.TakeFocusOnClick").RefersToRange.Offset(lCount,  _
        0) = cControl.TakeFocusOnClick
        .Names("Userforms.Text").RefersToRange.Offset(lCount, 0) =  _
        cControl.Text
```

```
            .Names("Userforms.TextAlign").RefersToRange.Offset(lCount, 0) =  _
            cControl.TextAlign
            .Names("Userforms.TextColumn").RefersToRange.Offset(lCount, 0) =  _
            cControl.TextColumn
            .Names("Userforms.Title").RefersToRange.Offset(lCount, 0) =  _
            cControl.Title
            .Names("Userforms.Top").RefersToRange.Offset(lCount, 0) =  _
            cControl.Top
            .Names("Userforms.TopIndex").RefersToRange.Offset(lCount, 0) =  _
            cControl.TopIndex
            .Names("Userforms.TripleState").RefersToRange.Offset(lCount, 0) =  _
            cControl.TripleState
            .Names("Userforms.Value").RefersToRange.Offset(lCount, 0) =  _
            cControl.Value
            .Names("Userforms.VerticalScrollbarSide").RefersToRange. _
            Offset(lCount, 0) = cControl.VerticalScrollBarSide
            .Names("Userforms.Visible").RefersToRange.Offset(lCount, 0) =  _
            cControl.Visible
            oSh.Names("Userforms.Width").RefersToRange.Offset(lCount, 0) =  _
            cControl.Width
            If cControl.Type = vbext_ct_MSForm Then
                .Names("Userforms.Code").RefersToRange.Offset(lCount, 0) =  _
                cControl.CodeModule.Lines(1, cControl.CodeModule.CountOfLines)
            End If
        End With
    End Sub

```

--- RenameComps ---

```
Private Sub Image2_MouseDown(ByVal Button As Integer, ByVal Shift As  _
Integer, ByVal X As Single, ByVal Y As Single)
    uDEV.Show
End Sub

Private Sub RenameComponents_Click()
    If pmWorkbook Is Nothing Then Set pmWorkbook = ActiveWorkbook
    Dim NewNames As Variant
    Dim i As Long
    NewNames = Split(textboxNewName, vbNewLine)
    For i = 0 To UBound(NewNames)
        If NewNames(i) = vbNullString Then
            NewNames(i) = LRenameListbox.List(i)
        End If
    Next i
    For i = 0 To UBound(NewNames)        ' RenameComps.LRenameListbox. _
    ListCount - 1
continue:
        On Error GoTo EH
        Select Case LRenameListbox.List(i, 0)
            'rename component
            Case Is = "Module", "Class", "UserForm"
                If LRenameListbox.List(i, 1) <> NewNames(i) Then
                    pmWorkbook.VBProject.VBComponents(LRenameListbox. _
                    List(i, 1)).Name = NewNames(i)
                End If
            'rename document (new worksheet name)
            Case Is = "Document"
                If LRenameListbox.List(i, 1) <> NewNames(i) Then
                    pmWorkbook.Sheets(LRenameListbox.List(i, 1)).Name =  _
                    NewNames(i)
                End If
        End Select
    Next
    'replace old names with new names in listbox
    For i = 0 To LRenameListbox.ListCount - 1
        LRenameListbox.List(i, 1) = NewNames(i)
    Next i
    'update user's new names in textbox with actual new names
    textboxNewName.Text = vbNullString
    Dim str As String
    str = Join(NewNames, vbNewLine)
    textboxNewName.Text = str
    MsgBox "Components renamed"
    Exit Sub
EH:
    'hanlde user giving duplicate name by incrementing
    NewNames(i) = NewNames(i) & i + 1
    Resume continue
End Sub
```

```
Private Sub UserForm_Initialize()
    If pmWorkbook Is Nothing Then Set pmWorkbook = ActiveWorkbook
    Dim vbComp As VBComponent
    For Each vbComp In pmWorkbook.VBProject.VBComponents
        If vbComp.Name <> "ThisWorkbook" Then
            LRenameListbox.AddItem
            LRenameListbox.List(LRenameListbox.ListCount - 1, 0) =  _
            ComponentTypeToString(vbComp.Type)
            If vbComp.Type <> vbext_ct_Document Then
                LRenameListbox.List(LRenameListbox.ListCount - 1, 1) =  _
                vbComp.Name
            Else
                LRenameListbox.List(LRenameListbox.ListCount - 1, 1) =  _
                GetSheetByCodeName(pmWorkbook, vbComp.Name).Name
            End If
        End If
    Next
    SortListboxOnColumn LRenameListbox, 0
    Dim str As String
    str = LRenameListbox.List(0, 1)
    For i = 1 To LRenameListbox.ListCount - 1
        str = str & vbNewLine & LRenameListbox.List(i, 1)
    Next
    textboxNewName.Text = str
End Sub
```

--- uProjectManager ---

```
    'images
    'Similar to jaslake,https://www.excelforum.com/excel-programming-vba-m
    'acros/1202015-print-userform-to-pdf-and-then-attach-it-to-an-email.html
    Private Const VK_SNAPSHOT = 44
    Private Const VK_LMENU = 164
    Private Const KEYEVENTF_KEYUP = 2
    Private Const KEYEVENTF_EXTENDEDKEY = 1
    'code printer
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Author    Anastasiou Alex
    ' Project   CodePrinter
    ' Purpose   Export active project's code as PDF. Code bl
    'ocks linked by shape. Keywords colored. Oddlines colored.
    ' Website   https://github.com/alexofrhodes
    ' Copyright MIT License 2021 Anastasiou Alex
    '
    ' Required References
    '   - Microsoft Visual Basic for Application Extensibility
    '   - mscorlib.dll (obsolete?)
    '
    ' Revision History:
    ' #  yyyy-mm-dd  COMMENTS
    ' 1  2021-08-05  Initial Release
    '
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Public PrintFileName As String
    Public Found1 As String
    Public fFound2 As String
    Dim rng As Range
    Public cell As Range
    Public s As Shape
    Public counter As Long
    Dim mafChrWid(32 To 127) As Double
    Dim msFontName As String

    Public Function PrintProject(wb As Workbook)
        Dim workbookName As String
        '    Dim ModuleName As String
        Dim Procedure As String
        workbookName = wb.Name        'ActiveProjName
        If ProtectedVBProject(wb) = True Or HasProject(wb) = False Then
            MsgBox "Project Empty or Protected"
            Exit Function
        End If
        'ThisWorkbook.Application.Visible = False
        'ThisWorkbook.IsAddin = False
        '    ModuleName = ActiveModule.Name
        Dim vbComp As VBComponent
        ResetPrinter
        Dim tmpString As Variant
```

```
    Dim i As Long
    Dim Procedures As Collection
    Set Procedures = New Collection
    Dim ws As Worksheet
    Dim wsName As String
    'Table of contents
    Procedures.Add "--- Table Of Contents ---" & vbNewLine & vbNewLine
    'document
    For Each vbComp In wb.VBProject.VBComponents
        If vbComp.Type = vbext_ct_Document Then
            For Each ws In wb.Worksheets
                If ws.CodeName = vbComp.Name Then wsName = ws.Name
            Next ws
            If vbComp.Name <> "ThisWorkbook" Then
                Procedures.Add "(" & ComponentTypeToString(vbComp.Type) &  _
                ")" & " " & wsName & " - " & vbComp.Name
            Else
                Procedures.Add "(" & ComponentTypeToString(vbComp.Type) &  _
                ")" & " " & vbComp.Name
            End If
            wsName = ""
        End If
    Next vbComp
    'class
    For Each vbComp In wb.VBProject.VBComponents
        If vbComp.Type = vbext_ct_ClassModule Then
            Procedures.Add "(" & ComponentTypeToString(vbComp.Type) & ")"  _
            & " " & vbComp.Name
        End If
    Next vbComp
    'module
    For Each vbComp In wb.VBProject.VBComponents
        If vbComp.Type = vbext_ct_StdModule Then
            Procedures.Add "(" & ComponentTypeToString(vbComp.Type) & ")"  _
            & " " & vbComp.Name
        End If
    Next vbComp
    'userform
    For Each vbComp In wb.VBProject.VBComponents
        If vbComp.Type = vbext_ct_MSForm Then
            Procedures.Add "(" & ComponentTypeToString(vbComp.Type) & ")"  _
            & " " & vbComp.Name
        End If
    Next vbComp
    'Code of components
    'document
    For Each vbComp In wb.VBProject.VBComponents
        If vbComp.Type = vbext_ct_Document Then
            'get sheet name
            For Each ws In wb.Worksheets
                If ws.CodeName = vbComp.Name Then wsName = ws.Name
            Next ws
```

```
            If vbComp.Name <> "ThisWorkbook" Then
                Procedures.Add "--- " & wsName & " - " & vbComp.Name & " - _
                --"
            Else
                Procedures.Add "--- " & vbComp.Name & " ---"
            End If
            wsName = ""
            If vbComp.CodeModule.CountOfLines > 0 Then
                tmpString = Split(GetCompText(vbComp), vbNewLine)
                For i = LBound(tmpString) To UBound(tmpString)
                    Procedures.Add " " & tmpString(i)
                Next i
            End If
        End If
    Next vbComp
    'class
    For Each vbComp In wb.VBProject.VBComponents
        If vbComp.Type = vbext_ct_ClassModule Then
            Procedures.Add "--- " & vbComp.Name & " ---"
            If vbComp.CodeModule.CountOfLines > 0 Then
                tmpString = Split(GetCompText(vbComp), vbNewLine)
                For i = LBound(tmpString) To UBound(tmpString)
                    Procedures.Add " " & tmpString(i)
                Next i
            End If
        End If
    Next vbComp
    'module
    For Each vbComp In wb.VBProject.VBComponents
        If vbComp.Type = vbext_ct_StdModule Then
            Procedures.Add "--- " & vbComp.Name & " ---"
            If vbComp.CodeModule.CountOfLines > 0 Then
                tmpString = Split(GetCompText(vbComp), vbNewLine)
                For i = LBound(tmpString) To UBound(tmpString)
                    Procedures.Add " " & tmpString(i)
                Next i
            End If
        End If
    Next vbComp
    'userform
    For Each vbComp In wb.VBProject.VBComponents
        If vbComp.Type = vbext_ct_MSForm Then
            Procedures.Add "--- " & vbComp.Name & " ---"
            If vbComp.CodeModule.CountOfLines > 0 Then
                tmpString = Split(GetCompText(vbComp), vbNewLine)
                For i = LBound(tmpString) To UBound(tmpString)
                    Procedures.Add " " & tmpString(i)
                Next i
            End If
        End If
    Next vbComp
    tmpString = CollectionToArray(Procedures)
```

```
    ThisWorkbook.Sheets("ProjectManagerPrinter").Range("B1:B" &  _
    UBound(tmpString) + 1).Value = WorksheetFunction.Transpose(tmpString)
    If CodePrinter = False Then GoTo ErrorHandler
    PrintPDF
ErrorHandler:
    'ThisWorkbook.IsAddin = True
    'ThisWorkbook.Application.Visible = True
End Function

Function CodePrinter() As Boolean
    ThisWorkbook.Sheets("ProjectManagerPrinter").Cells.Font.Name =  _
    "Consolas"
    RemoveBreaks
    BreakText
    NumberLinesPrinter
    ChgTxtColor
    GreenifyComments
    BoldPrinterComponents
    If findPairs = False Then
        CodePrinter = False
        Exit Function
    End If
    PrinterPageSetup
    ThisWorkbook.Sheets("ProjectManagerPrinter").rows(1).EntireRow.Insert
    copyLogoPrinter
    ShapesCompareLeft
    PageBreaksInPrinter
    CodePrinter = True
End Function

Function findPairs() As Boolean
    Dim ShapeTypeNumber As Long
    ShapeTypeNumber = 29
    Dim CloseTXT As String
    Dim X As Variant
    Dim ws As Worksheet
    Set ws = ThisWorkbook.Sheets("ProjectManagerPrinter")
    Dim trimCell As String
    Dim counter As Long
    For Each cell In ThisWorkbook.Sheets("ProjectManagerPrinter"). _
    Range("B:B").SpecialCells(xlCellTypeConstants)
        trimCell = Trim(cell.Text)
        If IsBlockStart(trimCell) Then
            Select Case openPair(trimCell)
                Case Is = "Case", "Else"
                    GoTo Skip
                Case Is = "If", "#If"
                    If Right(trimCell, 4) = "Then" Then        'Or  _
                    Right(trimCell, 1) = "_" Then
                        'ok
                    Else
                        GoTo Skip
```

```
                End If
            Case Is = "skip"
                GoTo Skip
            Case Else
                '
        End Select
        CloseTXT = closePair(trimCell)
        counter = Len(cell) - Len(trimCell)
        Found1 = cell.Address
        If FOUND2FOUND(ws, WorksheetFunction.Rept(" ", counter) &  _
        CloseTXT) = False Then
            GoTo Skip
            '                MsgBox "C
            'ode not properly indented." & vbNewLine & _
            '                       "Error
            'with closing pair of " & vbNewLine & cell.Text
            '                findPairs = False
            '                Exit Function
        End If
        found2 = ws.Range("B1:B" & ws.Cells(rows.Count, 2).End(xlUp). _
        Row) _
        .Find(WorksheetFunction.Rept(" ", counter) & CloseTXT & "*",  _
        After:=cell, LookAt:=xlWhole).Address
        X = StrWidth(Application.WorksheetFunction.Rept("A", counter), _
        "Consolas", 11)
        ws.Shapes.AddShape ShapeTypeNumber, ws.Range(Found1).Left + X  _
        - 10, ws.Range(Found1).Top + (cell.Height / 2), 5,  _
        Range(Found1, found2).Height - cell.Height
    End If
Skip:
    Next cell
    findPairs = True
End Function

Function FOUND2FOUND(ws As Worksheet, str As String) As Boolean
    FOUND2FOUND = True
    Dim tmp As Range
    Set tmp = ws.Range("B1:B" & ws.Cells(rows.Count, 2).End(xlUp).Row) _
    .Find(str & "*", After:=cell, LookAt:=xlWhole)
    If tmp Is Nothing Then FOUND2FOUND = False
End Function

Function openPair(strLine As String) As String
    Dim nPos As Integer
    Dim strTemp As String
    strTemp = Trim(strLine)
    nPos = InStr(1, strTemp, " ") - 1
    If nPos < 0 Then nPos = Len(strLine)
    strTemp = Left$(strLine, nPos)
    Select Case strTemp
        Case Is = "Private", "Public"
            strTemp = Trim(strLine)
```

```
            strTemp = Replace(strTemp, "Private ", "")
            strTemp = Replace(strTemp, "Public ", "")
            nPos = InStr(1, strTemp, " ") - 1
            If nPos < 0 Then nPos = Len(strTemp)
            strTemp = Left$(strTemp, nPos)
            If strTemp = "Function" Then
                openPair = "Function"
            ElseIf strTemp = "Sub" Then
                openPair = "Sub"
            Else
                GoTo Skip
            End If
        Case Is = "With"
            openPair = "With"
        Case Is = "For"
            openPair = "For"
        Case Is = "Do"
            openPair = "Do"
        Case Is = "While"
            openPair = "While"
        Case Is = "Select"
            openPair = "Select"
        Case Is = "Case"
            openPair = "Case"
        Case Is = "Sub"
            openPair = "Sub"
        Case Is = "Function"
            openPair = "Function"
        Case Is = "Property"
            openPair = "Property"
        Case Is = "Enum"
            openPair = "Enum"
        Case Is = "Type"
            openPair = "Type"
        Case "If", "#If"
            openPair = "If"
        Case "ElseIf", "#ElseIf", "Else", "Else:", "#Else", "#Else:"
            openPair = "Else"
        Case Else
    Skip:
            openPair = "skip"
    End Select
End Function

Function closePair(strLine As String) As String
    Dim nPos As Integer
    Dim strTemp As String
    nPos = InStr(1, strLine, " ") - 1
    If nPos < 0 Then nPos = Len(strLine)
    strTemp = Left$(strLine, nPos)
    Select Case strTemp
        Case Is = "Private", "Public"
```

```
            strTemp = Trim(strLine)
            strTemp = Replace(strTemp, "Private ", "")
            strTemp = Replace(strTemp, "Public ", "")
            nPos = InStr(1, strTemp, " ") - 1
            If nPos < 0 Then nPos = Len(strTemp)
            strTemp = Left$(strTemp, nPos)
            If strTemp = "Function" Then
                closePair = "End Function"
            ElseIf strTemp = "Sub" Then
                closePair = "End Sub"
            Else
                '
            End If
        Case Is = "With"
            closePair = "End With"
        Case Is = "For"
            closePair = "Next"
        Case Is = "Do", "While"
            closePair = "Loop"
        Case Is = "Select"        ', "Case"
            closePair = "End Select"
        Case Is = "Sub"
            closePair = "End Sub"
        Case Is = "Function"
            closePair = "End Function"
        Case Is = "Property"
            closePair = "End Property"
        Case Is = "Enum"
            closePair = "End Enum"
        Case Is = "Type"
            closePair = "End Type"
        Case "If", "#If", "ElseIf", "#ElseIf", "Else", "Else:", "#Else",  _
        "#Else:"
            closePair = "End If"
        Case Else
            '
    End Select
End Function

Sub PageBreaksInPrinter()
    ThisWorkbook.Sheets("ProjectManagerPrinter").ResetAllPageBreaks
    Dim rng As Range
    Set rng = Nothing
    Dim cell As Range
    With ThisWorkbook.Sheets("ProjectManagerPrinter")
        For Each cell In .Range("B1:B" & .Range("B" & .rows.Count). _
        End(xlUp).Row)
            If Left(Trim(cell.Value), 3) = "---" Then
                If rng Is Nothing Then
                    Set rng = cell
                Else
                    Set rng = Union(rng, cell)
```

```
                End If
            End If
        Next
        For Each cell In rng
            .HPageBreaks.Add Before:=.rows(cell.Row)
            .rows(cell.Row).PageBreak = xlPageBreakManual
        Next
    End With
End Sub

Sub FormatColourFormatters()
    Dim ws As Worksheet
    Set ws = ThisWorkbook.Sheets("ProjectManagerTXTColour")
    LBLcolourCode.ForeColor = ws.Range("J1").Value
    LBLcolourKey.ForeColor = ws.Range("J3").Value
    LBLcolourOdd.BackColor = ws.Range("J2").Value
    LBLcolourComment.ForeColor = ws.Range("J4").Value
End Sub

Sub ColorPaletteDialog(rng As Range, Lbl As MSForms.Label)
    If Application.Dialogs(xlDialogEditColor).Show(10, 0, 125, 125) =  _
    True Then
        'user pressed OK
        lcolor = ActiveWorkbook.Colors(10)
        rng.Value = lcolor
        rng.Offset(0, 1).Interior.Color = lcolor
        Lbl.ForeColor = lcolor
    End If
    ActiveWorkbook.ResetColors
End Sub

Sub RemoveBreaks()
    'remove line break loop
    Dim cell As Range
    Dim rng As Range
    With ThisWorkbook.Sheets("ProjectManagerPrinter")
        Set rng = .Range("B1:B" & .Range("B" & rows.Count).End(xlUp).Row)
    End With
    Dim coll As Collection
    Set coll = New Collection
    For Each cell In rng
        coll.Add CleanTrim(cell.Value)
    Next cell
    Dim arr As Variant
    arr = CollectionToArray(coll)
    rng.Value = WorksheetFunction.Transpose(arr)
End Sub

Function CleanTrim(ByVal s As String, Optional ConvertNonBreakingSpace As  _
Boolean = True) As String
    'remove line break function
    Dim X As Long, CodesToClean As Variant
```

```
    CodesToClean = Array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, _
    15, 16, 17, 18, 19, 20, _
    21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 127, 129, 141, 143, 144,  _
    157)
    If ConvertNonBreakingSpace Then s = Replace(s, Chr(160), " ")
    For X = LBound(CodesToClean) To UBound(CodesToClean)
        If InStr(s, Chr(CodesToClean(X))) Then
            s = Replace(s, Chr(CodesToClean(X)), vbNullString)
        End If
    Next
    CleanTrim = s
    '  CleanTrim = WorksheetFunction.Trim(S)
End Function

Sub GreenifyComments()
    Dim cell As Range
    Dim sh As Worksheet
    Set ws = ThisWorkbook.Sheets("ProjectManagerPrinter")
    Set rng = Nothing
    For Each cell In ws.Range("B1:B" & ws.Range("B" & rows.Count). _
    End(xlUp).Row)
        If Left(Trim(cell.Value), 1) = "'" Or Left(Trim(cell.Value), 3) =  _
        "Rem" Then
            cell.Font.Color = ThisWorkbook. _
            Sheets("ProjectManagerTXTColour").Range("J4").Value
        End If
    Next
End Sub

Sub SpaceProcsInPrinter()
    'add empty line between end of sub/fun and start of next
    Dim cell As Range
    Dim rng As Range
    With ThisWorkbook.Sheets("ProjectManagerPrinter")
        Set rng = .Range("B2:B" & .Range("B" & rows.Count).End(xlUp).Row)
    End With
    With rng
        Set cell = .Find("*End Sub", LookIn:=xlValues)
        If Not cell Is Nothing Then
            firstAddress = cell.Address
            Do
                With cell.Borders(xlEdgeBottom)
                    .LineStyle = xlContinuous
                    .Weight = xlThin
                    .Color = vbBlack
                End With
                Set cell = .FindNext(cell)
            Loop While Not cell Is Nothing And cell.Address <>  _
            firstAddress
        End If
    End With
    With rng
```

```
        Set cell = .Find("*End Function", LookIn:=xlValues)
        If Not cell Is Nothing Then
            firstAddress = cell.Address
            Do
                With cell.Borders(xlEdgeBottom)
                    .LineStyle = xlContinuous
                    .Weight = xlThin
                End With
                Set cell = .FindNext(cell)
            Loop While Not cell Is Nothing And cell.Address <>  _
            firstAddress
        End If
    End With
End Sub

Sub NumberLinesPrinter()
    Dim lRow
    Dim cell As Range
    With ThisWorkbook.Sheets("ProjectManagerPrinter")
        lRow = .Range("B" & .rows.Count).End(xlUp).Row
        For Each cell In .Range("B1:B" & lRow)
            If cell.Row Mod 2 = 0 Then
                Range(cell.Offset(0, -1), cell.Offset(0, 1)).Interior. _
                Color = _
                ThisWorkbook.Sheets("ProjectManagerTXTColour").Range("J2") _
                .Value
            End If
        Next cell
        '.Columns(1).HorizontalAlignment = xlLeft
    End With
End Sub

Sub BoldPrinterComponents()
    'format printer lines with component names
    Dim rng As Range
    Set rng = Nothing
    Dim cell As Range
    With ThisWorkbook.Sheets("ProjectManagerPrinter")
        For Each cell In .Range("B1:B" & .Range("B" & .rows.Count). _
        End(xlUp).Row)
            If Left(Trim(cell.Value), 3) = "---" Then
                If rng Is Nothing Then
                    Set rng = cell
                Else
                    Set rng = Union(rng, cell)
                End If
            End If
        Next
    End With
    If rng Is Nothing Then Exit Sub
    rng.Font.size = 18
    rng.Font.Bold = True
```

```
        rng.Font.Color = vbBlack
    End Sub

    Sub copyLogoPrinter()
        ThisWorkbook.Sheets("ProjectManagerSettings").Shapes("LOGO").Copy
        ThisWorkbook.Sheets("ProjectManagerPrinter").Paste ThisWorkbook. _
        Sheets("ProjectManagerPrinter").Range("B1")
        Dim shp As Shape
        Set shp = ThisWorkbook.Sheets("ProjectManagerPrinter").Shapes("LOGO")
        With ThisWorkbook.Sheets("ProjectManagerPrinter")
            shp.Left = .Range("B1").Left + ((.Range("B1").Width - shp.Width) / _
            2)
            shp.Top = .Range("B1").Top
            .rows(1).RowHeight = shp.Height + 50
            .Range("A2:C2").Interior.ColorIndex = 0
            With .Range("B1")
                .HorizontalAlignment = xlCenter
                .VerticalAlignment = xlVAlignBottom
                .Value = vbNewLine & vbNewLine & pmWorkbook.Name & vbNewLine  _
                & "www.github.com/alexofrhodes"
                .Characters.Font.size = 18
                .Characters.Font.Bold = True
                .Characters.Font.Underline = False
                .Characters.Font.ColorIndex = 10
                .Characters.Font.Name = "Comic Sans MS"
            End With
        End With
    End Sub

    Sub PrinterPageSetup()
        With ThisWorkbook.Sheets("ProjectManagerPrinter").PageSetup
            'narrow margins
            .LeftMargin = Application.InchesToPoints(0.25)
            .RightMargin = Application.InchesToPoints(0.25)
            .TopMargin = Application.InchesToPoints(0.25)
            .BottomMargin = Application.InchesToPoints(0.75)
            'left footer filename
            Dim FileName As String
            FileName = PrintFileName
            .LeftFooter = FileName
            '.LeftFooter =  "&F"    'Filename?
            'center footer page of pages
            .CenterFooter = "Page &P of &N"
            'right footer date
            .RightFooter = "&D"
            'fit all columns in one page width
            .FitToPagesWide = 1
            .FitToPagesTall = False
        End With
    End Sub

    Sub ShapesCompareLeft()
```

```
    'if code block connector lines spill to the next page,
    'we can easily follow the one we want if each line has it's own colour
    Dim rnd As Long
    Dim n As Variant
    Dim i As Long
    Dim s As Shape
    Dim sNames
    Set sNames = CreateObject("System.Collections.ArrayList")
    'rename lines to their .left position
    For Each s In ThisWorkbook.Sheets("ProjectManagerPrinter").Shapes
        If s.Name <> "LOGO" Then
            s.Name = s.Left
            'create a unique array of names
            If Not sNames.CONTAINS(s.Name) Then
                sNames.Add s.Name
            End If
        End If
    Next s
    'assign unique colour to lines by level (left)
    For Each n In sNames
        rnd = RandomRGB
        For Each s In ThisWorkbook.Sheets("ProjectManagerPrinter").Shapes
            If s.Name <> "LOGO" Then
                If s.Name = n Then
                    With s.line
                        .ForeColor.RGB = rnd
                        .Weight = 1.5
                    End With
                End If
            End If
        Next s
    Next n
    Set sNames = Nothing
End Sub

Function RandomRGB()
    RandomRGB = RGB(Int(rnd() * 255), Int(rnd() * 255), Int(rnd() * 255))
End Function

Function StrWidth(s As String, sFontName As String, fFontSize As Double)  _
As Double
    ' Returns the approximate width in points of a text string
    ' in a specified font name and font size
    ' Does not account for kerning
    Dim i As Long
    Dim j As Long
    If Len(sFontName) = 0 Then
        Exit Function
    End If
    If sFontName <> msFontName Then
        If Not InitChrWidths(sFontName) Then
            Exit Function
```

```
        End If
    End If
    For i = 1 To Len(s)
        j = Asc(Mid(s, i, 1))
        If j >= 32 Then
            StrWidth = StrWidth + fFontSize * mafChrWid(j)
        End If
    Next i
End Function

Function InitChrWidths(sFontName As String) As Boolean
    Dim i As Long
    Select Case sFontName
        Case "Consolas"
            For i = 32 To 127
                Select Case i
                    Case 32 To 127
                        mafChrWid(i) = 0.5634
                End Select
            Next i
            '    Case "Arial"
            '        For i = 32 To 127
            '            Select Case i
            '            Case 39, 106, 108
            '                mafChrWid(i) = 0.1902
            '            Case 105, 116
            '                mafChrWid(i) = 0.2526
            '            Case 32, 33, 4
            '4, 46, 47, 58, 59, 73, 91 To 93, 102, 124
            '                mafChrWid(i) = 0.3144
            '            Case 34, 40, 41, 45, 96, 114, 123, 125
            '                mafChrWid(i) = 0.3768
            '            Case 42, 94, 118, 120
            '                mafChrWid(i) = 0.4392
            '            Case 107, 115, 122
            '                mafChrWid(i) = 0.501
            '            Case 35, 36, 48 To 57, 63, 74,
            '76, 84, 90, 95, 97 To 101, 103, 104, 110 To 113, 117, 121
            '                mafChrWid(i) = 0.5634
            '            Case 43, 60 To 62, 70, 126
            '                mafChrWid(i) = 0.6252
            '            Case 38, 65, 66, 69
            ', 72, 75, 78, 80, 82, 83, 85, 86, 88, 89, 119
            '                mafChrWid(i) = 0.6876
            '            Case 67, 68, 71, 79, 81
            '                mafChrWid(i) = 0.7494
            '            Case 77, 109, 127
            '                mafChrWid(i) = 0.8118
            '            Case 37
            '                mafChrWid(i) = 0.936
            '            Case 64, 87
            '                mafChrWid(i) = 1.0602
```

```
'            End Select
'        Next i
'
'    Case "Calibri"
'        For i = 32 To 127
'            Select Case i
'            Case 32, 39, 44, 46, 73, 105, 106, 108
'                mafChrWid(i) = 0.2526
'            Case 40, 41, 4
'5, 58, 59, 74, 91, 93, 96, 102, 123, 125
'                mafChrWid(i) = 0.3144
'            Case 33, 114, 116
'                mafChrWid(i) = 0.3768
'            Case 34, 47, 76, 92, 99, 115, 120, 122
'                mafChrWid(i) = 0.4392
'            Case 35, 42, 43, 60 To 63, 69, 70,
'83, 84, 89, 90, 94, 95, 97, 101, 103, 107, 118, 121, 124, 126
'                mafChrWid(i) = 0.501
'            Case 36, 48 To 57, 66, 67,
'75, 80, 82, 88, 98, 100, 104, 110 To 113, 117, 127
'                mafChrWid(i) = 0.5634
'            Case 65, 68, 86
'                mafChrWid(i) = 0.6252
'            Case 71, 72, 78, 79, 81, 85
'                mafChrWid(i) = 0.6876
'            Case 37, 38, 119
'                mafChrWid(i) = 0.7494
'            Case 109
'                mafChrWid(i) = 0.8742
'            Case 64, 77, 87
'                mafChrWid(i) = 0.936
'            End Select
'        Next i
'    Case "Tahoma"
'        For i = 32 To 127
'            Select Case i
'            Case 39, 105, 108
'                mafChrWid(i) = 0.2526
'            Case 32, 44, 46, 102, 106
'                mafChrWid(i) = 0.3144
'            Case 33, 45, 58, 59, 73, 114, 116
'                mafChrWid(i) = 0.3768
'            Case 34, 40, 41, 47, 74, 91 To 93, 124
'                mafChrWid(i) = 0.4392
'            Case 63, 76, 99, 107, 115, 118, 120 To 123, 125
'                mafChrWid(i) = 0.501
'            Case 36, 42, 48 To 57, 70,
'80, 83, 95 To 98, 100, 101, 103, 104, 110 To 113, 117
'                mafChrWid(i) = 0.5634
'            Case 66, 67, 69, 75, 84, 86, 88, 89, 90
'                mafChrWid(i) = 0.6252
'            Case 38, 65, 71, 72, 78, 82, 85
```

```
'                mafChrWid(i) = 0.6876
'            Case 35, 43, 60 To 62, 68, 79, 81, 94, 126
'                mafChrWid(i) = 0.7494
'            Case 77, 119
'                mafChrWid(i) = 0.8118
'            Case 109
'                mafChrWid(i) = 0.8742
'            Case 64, 87
'                mafChrWid(i) = 0.936
'            Case 37, 127
'                mafChrWid(i) = 1.0602
'            End Select
'        Next i
'    Case "Lucida Console"
'        For i = 32 To 127
'            Select Case i
'            Case 32 To 127
'                mafChrWid(i) = 0.6252
'            End Select
'        Next i
'
'    Case "Times New Roman"
'        For i = 32 To 127
'            Select Case i
'            Case 39, 124
'                mafChrWid(i) = 0.1902
'            Case 32, 44, 46, 59
'                mafChrWid(i) = 0.2526
'            Case 33, 34, 4
'7, 58, 73, 91 To 93, 105, 106, 108, 116
'                mafChrWid(i) = 0.3144
'            Case 40, 41, 45, 96, 102, 114
'                mafChrWid(i) = 0.3768
'            Case 63, 74, 97, 115, 118, 122
'                mafChrWid(i) = 0.4392
'            Case 94, 98 To 101, 103,
'104, 107, 110, 112, 113, 117, 120, 121, 123, 125
'                mafChrWid(i) = 0.501
'            Case 35, 36,
'42, 48 To 57, 70, 83, 84, 95, 111, 126
'                mafChrWid(i) = 0.5634
'            Case 43, 60 To 62, 69, 76, 80, 90
'                mafChrWid(i) = 0.6252
'            Case 65 To 67, 82, 86, 89, 119
'                mafChrWid(i) = 0.6876
'            Case 68, 71, 72, 75, 78, 79, 81, 85, 88
'                mafChrWid(i) = 0.7494
'            Case 38, 109, 127
'                mafChrWid(i) = 0.8118
'            Case 37
'                mafChrWid(i) = 0.8742
'            Case 64, 77
```

```
            '                mafChrWid(i) = 0.936
            '            Case 87
            '                mafChrWid(i) = 0.9984
            '            End Select
            '        Next i
        Case Else
            MsgBox "Font name """ & sFontName & """ not available!",  _
            vbCritical, "StrWidth"
            Exit Function
    End Select
    msFontName = sFontName
    InitChrWidths = True
End Function

Public Sub ChgTxtColor()
    With ThisWorkbook.Sheets("ProjectManagerPrinter").Cells.Font
        .Color = ThisWorkbook.Sheets("ProjectManagerTXTColour"). _
        Range("J1").Value
        .FontStyle = "Normal"
    End With
    Dim rng As Range
    Set rng = ThisWorkbook.Sheets("ProjectManagerPrinter").UsedRange
    Dim cell As Range
    Dim NumChars As Long
    Dim StartChar As Long
    Dim cellChar As Long
    Dim EndWords As Long
    Dim keywords As Range
    On Error Resume Next
    For Each cell In rng
        cellChar = Len(cell)
        For Each keywords In ThisWorkbook. _
        Sheets("ProjectManagerTXTColour").Range("A1").CurrentRegion. _
        Offset(1).Resize(, 1).SpecialCells(xlCellTypeConstants)
            StartChar = InStrExact(1, cell.Text, keywords.Text)
            Do Until StartChar >= cellChar Or StartChar = 0
                NumChars = Len(keywords.Text)
                EndWords = StartChar + NumChars
                If Mid(cell.Text, StartChar - 1, 1) = " " Or StartChar =  _
                1 Then
                    If Mid(cell.Text, EndWords, 1) = " " Or EndWords >=  _
                    cellChar Then
                        With cell.Characters(Start:=StartChar,  _
                        Length:=NumChars).Font
                            'format matches
                            .FontStyle = "Bold"
                            .Color = ThisWorkbook. _
                            Sheets("ProjectManagerTXTColour").Range("J3"). _
                            Value
                        End With
                    End If
                End If
```

```
                StartChar = InStr(EndWords, cell.Text, keywords.Text)
            Loop
        Next
    Next
End Sub

Sub ResetPrinter(Optional keepText As Boolean = False)
    '    OptOn
    With ThisWorkbook.Sheets("ProjectManagerPrinter")
        .ResetAllPageBreaks
        If keepText = False Then
            .[A:C].Clear
        Else
            .[A:C].ClearFormats
            .Cells.Font.ColorIndex = vbBlack
            .Cells.Font.Bold = False
        End If
        .Columns("A:A").ColumnWidth = 3        '3
        .Columns("C:C").ColumnWidth = 1
        For Each s In ThisWorkbook.Sheets("ProjectManagerPrinter").Shapes
            'If Left(s.name, 2) <> "cp" Then
            s.Delete
            'End If
        Next
        .Cells.Font.Name = "Consolas"
        If .PageSetup.Orientation = xlPortrait Then
            .Columns("B:B").ColumnWidth = 90
        Else
            .Columns("B:B").ColumnWidth = 120
        End If
        .Cells.WrapText = False
        .Cells.UseStandardHeight = True
        '        .Cells.UseStandardWidth = True
    End With
    '    Application.ScreenUpdating = True
End Sub

Sub BreakText()
    'Coded by Anastasiou Alex
    'Version 1
    '20/1/2021
    '    Dim l As Long
    '    l = Timer()
    'to get things right, use a monospace font like Consolas
    Dim cell      As Range
    Dim TmpStr    As String
    Dim Splitter  As Integer
    Dim counter   As Integer
    Dim Limit     As Integer
    'how many characters fit your cell width (find manually)
    If ThisWorkbook.Sheets("ProjectManagerPrinter").PageSetup.Orientation  _
    = xlPortrait Then
```

```
            Limit = 75
        Else
            Limit = 100         '80
        End If
        'For which range to run
        Dim rng As Range
        With ThisWorkbook.Sheets("ProjectManagerPrinter")
            Set rng = .Range("B1:B" & .Range("B" & .rows.Count).End(xlUp).Row)
        End With
        Dim coll As Collection
        Set coll = New Collection
        On Error Resume Next
        For Each cell In rng
            TmpStr = cell.Text
            'remove unnecessary spaces (not trimming)
            If Right(cell.Offset(-1, 0), 1) = "_" Then
                counter = Len(cell.Offset(-1, 0)) - Len(Trim(cell.Offset(-1,  _
                0)))
                TmpStr = Application.WorksheetFunction.Rept(" ", counter) &  _
                Trim(cell.Text)
                cell.Value = TmpStr
            End If
            'create collection
            'if len of cell text <= limit then take as is
    REPEATME:
            If Len(TmpStr) > Limit Then
                counter = Len(TmpStr) - Len(Trim(TmpStr))
                'if comment
                'BreakText and add first part to collection. Repeat
                If Left(Trim(TmpStr), 1) = "'" Or Left(Trim(TmpStr), 3) =  _
                "Rem" Then
                    Splitter = Len(cell) / 2
                    coll.Add Left(TmpStr, Splitter)         '& " _"
                    TmpStr = Application.WorksheetFunction.Rept(" ", counter)  _
                    & _
                    "'" & Trim(Mid(TmpStr, Splitter + 1))
                    GoTo REPEATME
                    'if not comment
                Else
                    'find which symbol is closest to the limit and before it
                    Splitter = InStrRev(TmpStr, WhichFirst(TmpStr, ".`,`/`-`  _
                    `)", "`", Limit), Limit)
                    coll.Add Left(TmpStr, Splitter) & " _"
                    TmpStr = Application.WorksheetFunction.Rept(" ", counter)  _
                    & _
                    Trim(Mid(TmpStr, Splitter + 1))
                    GoTo REPEATME
                End If
            Else
                coll.Add (TmpStr)
            End If
        Next cell
```

```
    'replace sheet printer cells with broken text from collection
    Dim arr
    arr = CollectionToArray(coll)
    With ThisWorkbook.Sheets("ProjectManagerPrinter")
        .Cells.Clear
        .Range("B1:B" & UBound(arr) + 1).Value = WorksheetFunction. _
        Transpose(arr)
        .Cells.Font.Name = "Consolas"
    End With
    '    Debug.Print Timer() - l
End Sub

Sub testWhichFirst()
    If ActiveCell = vbNullString Then
        Exit Sub
    End If
    WhichFirst ActiveCell, ".`,`/`-`_` `)", "`", Len(ActiveCell)
End Sub

Function WhichFirst(st As String, items As String, delim As String,  _
AfterPosition As Integer)
    'Coded by Anastasiou Alex
    'Version 1
    '20/1/2021
    '
    'PARAMETERS
    'st : which string to parse
    'items : which characters are we looking for
    'delim : delimeter to split passed items
    'AfterPosition :
    Dim i As Long
    Dim varr As Variant
    varr = Split(items, delim)
lp:
    On Error Resume Next
    'WhichFirst set to last varr item so it will be looped again?
    WhichFirst = varr(UBound(varr))
    For i = LBound(varr) To UBound(varr)
        'Debug.Print varr(i) & InStrRev(st, varr(i), AfterPosition)
        'find the item closest to the limit
        If InStrRev(st, varr(i), AfterPosition) > InStrRev(st, WhichFirst, _
        AfterPosition) Then
            WhichFirst = varr(i)
        End If
    Next i
    '    Debug.Print "Limit", AfterPosition & vbNewLine & _
    "Closest Item", WhichFirst & vbNewLine & _
    "Found At", InStrRev(st, WhichFirst, AfterPosition)
End Function

Function HasProject(wb As Workbook) As Boolean
    Dim WbProjComp As Object
```

```
    On Error Resume Next
    Set WbProjComp = wb.VBProject.VBComponents
    If Not WbProjComp Is Nothing Then HasProject = True
End Function

Sub OptOn()
    Application.ScreenUpdating = False
    Application.DisplayStatusBar = False
    Application.Calculation = xlCalculationManual
    Application.EnableEvents = False
    ' Note: this is a sheet-level setting.
    ActiveSheet.DisplayPageBreaks = False
End Sub

Sub OptOff()
    Application.ScreenUpdating = True
    Application.DisplayStatusBar = True
    Application.Calculation = xlCalculationAutomatic
    Application.EnableEvents = True
    ' Note: this is a sheet-level setting.
    ActiveSheet.DisplayPageBreaks = False
End Sub

Public Function ActiveProjName() As String
    'name of active project in vbeditor
    ActiveProjName = Mid(Application.VBE.ActiveVBProject.FileName,  _
    InStrRev(Application.VBE.ActiveVBProject.FileName, "\") + 1)
End Function

Sub PrintPDF()
    Dim FilePath As String
    FilePath = Environ("USERprofile") & "\Documents\" &  _
    "vbArc\CodePrinter\"
    Dim FileName As String
    FileName = Left(pmWorkbook.Name, InStr(1, pmWorkbook.Name, ".") - 1)
    Dim saveLocation As String
    saveLocation = FilePath        '& fileName & "\"
    If DIR(saveLocation, vbDirectory) = "" Then
        FoldersCreate saveLocation
    End If
    FilePath = saveLocation & FileName
    ThisWorkbook.Sheets("ProjectManagerPrinter").ExportAsFixedFormat _
    Type:=xlTypePDF, _
    FileName:=FilePath
    'FollowLink saveLocation
    'FollowLink filePath & ".pdf"
End Sub

Public Sub delay(seconds As Long)
    Dim endTime As Date
    endTime = DateAdd("s", seconds, Now())
    Do While Now() < endTime
```

```
        DoEvents
    Loop
End Sub

Rem images
Sub UserformToPDF(wb As Workbook, Path As String)
    Application.VBE.mainwindow.Visible = True
    Do While Application.VBE.mainwindow.Visible = False
        DoEvents
    Loop
    CloseVBEwindows
    Dim vbComp As VBComponent
    For Each vbComp In wb.VBProject.VBComponents
        If vbComp.Type = vbext_ct_MSForm Then
            vbComp.Activate
            DoEvents
            'Application.Wait (Now + TimeValue("0:00:3"))
            Call WindowToPDF(PathMaker(Path, vbComp.Name, "pdf"))
        End If
    Next
End Sub

Sub ExportWorksheetsToPDF(wb As Workbook, expPath As String)
    wb.Activate
    Dim ws As Worksheet
    For Each ws In wb.Worksheets
        Application.PrintCommunication = False
        With ws.PageSetup
            .FitToPagesWide = 1
            .FitToPagesTall = False
        End With
        Application.PrintCommunication = True
        If WorksheetFunction.CountA(ws.Cells) > 0 Then
            ws.ExportAsFixedFormat xlTypePDF, PathMaker(expPath, ws.Name,  _
            "pdf"), , True
        End If
    Next ws
End Sub

Function WindowToPDF(pdf$, Optional Orientation As Integer = xlLandscape,  _
_
Optional FitToPagesWide As Integer = 1) As Boolean
    Dim calc As Integer, ws As Worksheet
    With Application
        .ScreenUpdating = False
        .EnableEvents = False
        calc = .Calculation
        .Calculation = xlCalculationManual
    End With
    keybd_event VK_LMENU, 0, KEYEVENTF_EXTENDEDKEY, 0
    keybd_event VK_SNAPSHOT, 0, KEYEVENTF_EXTENDEDKEY, 0
    keybd_event VK_SNAPSHOT, 0, KEYEVENTF_EXTENDEDKEY + KEYEVENTF_KEYUP, 0
```

```
        keybd_event VK_LMENU, 0, KEYEVENTF_EXTENDEDKEY + KEYEVENTF_KEYUP, 0
        Set ws = Workbooks.Add(xlWBATWorksheet).Worksheets(1)
        Application.Wait (Now + TimeValue("0:00:1"))
        With ws
            .PasteSpecial Format:="Bitmap", link:=False, DisplayAsIcon:=False
            .Range("A1").Select
            .PageSetup.Orientation = Orientation
            .PageSetup.FitToPagesWide = FitToPagesWide
            .PageSetup.Zoom = False
            .ExportAsFixedFormat Type:=xlTypePDF, FileName:=pdf, _
            quality:=xlQualityStandard, IncludeDocProperties:=True, _
            IgnorePrintAreas:=False, OpenAfterPublish:=False
            .Parent.Close False
        End With
        With Application
            .ScreenUpdating = True
            .EnableEvents = True
            .Calculation = calc
            .CutCopyMode = False
        End With
        WindowToPDF = DIR(pdf) <> ""
    End Function

    Function PathMaker(wbPath As String, FileName As String, fileExtention As  _
    String) As String
        If Right(wbPath, 1) <> "\" Then wbPath = wbPath & "\"
        PathMaker = wbPath & FileName & "." & fileExtention
        Do While InStr(1, PathMaker, "..") > 0
            PathMaker = Replace(PathMaker, "..", ".")
        Loop
    End Function

    Rem imports
    Sub ImportComponents(wb As Workbook)
        Dim varr        'As Variant
        Dim element        'As String
        Dim proceed As Boolean, hasWorksheets As Boolean
        proceed = True
        Dim compName As String
        'file dialogue multiselect components to import
        varr = GetFilePath(Array("bas", "frm", "cls"), True)
        If Not IsArrayAllocated(varr) Then Exit Sub
        Dim vbproj As VBProject
        Set vbproj = wb.VBProject
        Dim coll As Collection
        Set coll = New Collection
        For Each element In varr
            compName = GetFilePartName(CStr(element), False)
            Debug.Print compName
            If compName Like "DocClass*" Then
                compName = Right(compName, Len(compName) - 6)
                hasWorksheets = True
```

```
        End If
        If ModuleExists(compName, wb) = True Then
            proceed = False
            coll.Add compName
        End If
    Next element
    If proceed = False Then GoTo ErrorHandler
    Dim wasOpen As Boolean
    Dim wbSource As Workbook
    Dim wbSourceName As String
    Dim basePath As String
    basePath = GetFilePartPath(varr(1), True)
    If hasWorksheets = True Then
        wbSourceName = DIR(basePath & "*.xl*")
        If wbSourceName <> "" Then
            wasOpen = WorkbookIsOpen(wbSourceName)
            If wasOpen = False Then
                Set wbSource = Workbooks.Open(basePath & wbSourceName)
            Else
                Set wbSource = Workbooks(wbSourceName)
            End If
        End If
    End If
    For Each element In varr
        compName = GetFilePartName(CStr(element), False)
        If Not compName Like "DocClass*" Then
            vbproj.VBComponents.Import element
        Else
            compName = Right(compName, Len(compName) - 9)
            If compName <> "ThisWorkbook" Then
                wbSource.Sheets(compName).Copy Before:=wb.Sheets(1)
            End If
        End If
    Next element
    GoTo exitHandler
ErrorHandler:
    Dim str As String
    str = "The following components already exist. All import canceled."
    For Each element In coll
        str = str & vbNewLine & element
    Next element
    MsgBox str
    Exit Sub
exitHandler:
    If wasOpen = False And WorkbookIsOpen(wbSourceName) Then wbSource. _
    Close False
    Set vbproj = Nothing
    Set coll = Nothing
    Set wbSource = Nothing
    MsgBox "Import successful"
End Sub
```

```
    Rem exports
Function ExportProject( _
wb As Workbook, _
Optional exportXML As Boolean, _
Optional ExportSheets As Boolean, _
Optional ExportForms As Boolean, _
Optional ExportFormsProperties As Boolean, _
Optional PrintCode As Boolean, _
Optional bSeparateProcedures As Boolean, _
Optional bMainExport As Boolean)
    Dim workbookCleanName    As String:             workbookCleanName =  _
    Left(wb.Name, InStrRev(wb.Name, ".") - 1)
    Dim workbookExtension       As String:              workbookExtension  _
    = Right(wb.Name, Len(wb.Name) - InStr(1, wb.Name, "."))
    Dim mainPath                        As String:              mainPath  _
    = Environ("USERprofile") & "\Documents\" & "vbArc\Code Library\"
    Dim exportPath                      As String:                _
    exportPath = mainPath & workbookCleanName & "\"
    exportPath = exportPath & Format(Now, "YYMMDD HHNNSS") & "\"
    'create folders
    FoldersCreate mainPath: FoldersCreate exportPath
    If bMainExport = True Then
        'export workbook backup
        wb.SaveCopyAs exportPath & wb.Name
        'export references
        ExportReferencesToConfigFile exportPath
        Dim procColl As Collection, Procedure As Variant, vbComp As  _
        VBComponent, Extension As String
        'export unified code to easily compare changes
        For Each vbComp In wb.VBProject.VBComponents
            TxtAppend exportPath & "#UnifiedProject.txt",  _
            GetCompText(vbComp)        'add comp's text to unified  _
            project's txt
        Next
        'Export Components
        For Each vbComp In wb.VBProject.VBComponents
            Select Case vbComp.Type
                Case vbext_ct_ClassModule, vbext_ct_Document:   Extension  _
                = ".cls"
                Case vbext_ct_MSForm:                                      _
                Extension = ".frm"
                Case vbext_ct_StdModule:                                   _
                Extension = ".bas"
                Case Else:                                                 _
                Extension = ".txt"
            End Select
            If vbComp.Type = vbext_ct_Document Then
                If vbComp.Name = "ThisWorkbook" Then
                    Rem @TODO change DocClass &
                    'vbComp.name to vbComp.name & extension (=.doccls)
                    vbComp.Export exportPath & "DocClass " & vbComp.Name  _
                    & Extension
```

```
                Else
                    vbComp.Export exportPath & "DocClass " &  _
                    GetSheetByCodeName(wb, vbComp.Name).Name & Extension
                End If
            Else
                'export component
                vbComp.Export exportPath & vbComp.Name & Extension
            End If
        Next
        'export declarations
        TxtAppend exportPath & "Declarations.txt",  _
        ArrayToString(CollectionsToArrayTable(getDeclarations(pmWorkbook)) _
        )
    End If
    'List userforms properties (by JKP)
    If chFormsProperties = True Then ListFormsExport wb, exportPath
    'Export procedures separately
    If bSeparateProcedures = True Then
        Dim ProcedurePath As String
        Dim ans As Long
        ans = MsgBox("If there are too many procedures the proccess will  _
        be slow. Proceed?", vbYesNo)
        If ans = vbYes Then
            For Each vbComp In wb.VBProject.VBComponents
                ProcedurePath = exportPath & vbComp.Name & " Procedures\"
                FoldersCreate ProcedurePath
                Set procColl = ProcListCollection(vbComp)
                'export component's procedures as txt
                For Each Procedure In procColl
                    TxtAppend ProcedurePath & Procedure & ".txt",  _
                    GetProcText(vbComp, CStr(Procedure))
                Next Procedure
            Next
        End If
    End If
    'Print to PDF, original feature (c
    'odeblocks linked, choose colour scheme)
    If PrintCode = True Then
        PrintFileName = wb.Name
        'IndentWorkbook wb
        PrintProject wb
    End If
    'export Userform To PDF
    If ExportForms = True Then
        If wb.Name <> ThisWorkbook.Name Then UserformToPDF wb, exportPath
    End If
    'Export Worksheets To Image
    If ExportSheets = True Then
        Dim EXT As String: EXT = Right(wb.Name, Len(wb.Name) - InStr(1,  _
        wb.Name, "."))
        If EXT = "xlam" Or EXT = "xla" Then wb.IsAddin = False
        ExportWorksheetsToPDF wb, exportPath
```

```
        If EXT = "xlam" Or EXT = "xla" Then wb.IsAddin = True
    End If
    'Export ribbon xml (by JKP)
    If exportXML = True Then
        Dim FullPath As String
        FullPath = wb.FullName
        wb.Close True
        ExtractRibbonX FullPath, exportPath & "customUI.xml"
        Workbooks.Open FullPath
    End If
    MsgBox "Export complete"
    'open export folder
    'FollowLink exportPath
End Function

Rem credit to Todar
Public Sub ExportReferencesToConfigFile(RefPath As String)
    Dim myProject As VBProject
    Set myProject = pmWorkbook.VBProject        'Application.VBE. _
    ActiveVBProject
    Dim fso As New Scripting.FileSystemObject
    With fso.OpenTextFile(RefPath & "References.Txt", ForWriting, True)
        Dim library As Reference
        For Each library In myProject.References
            .WriteLine library.Name & vbTab & library.GUID & vbTab &  _
            library.Major & vbTab & library.Minor
        Next
    End With
End Sub

Rem @TODO
Public Sub ImportReferencesFromConfigFile()
    Dim fso As New Scripting.FileSystemObject
    With fso.OpenTextFile(exportPath & "References.Txt", ForReading, True)
        Dim line As Long
        Do While Not .AtEndOfStream
            Dim values As Variant
            values = Split(.ReadLine, vbTab)
            ' Just skip if it already exists
            On Error Resume Next
            ThisWorkbook.VBProject.References.AddFromGuid values(1),  _
            values(2), values(3)
        Loop
    End With
End Sub

'refresh
''''''''''''''''''''''''''''''''''''''''''''
'Ron De Bruin Export - Import Modules START'
''''''''''''''''''''''''''''''''''''''''''''
Sub RefreshComponents(wkbSource As Workbook)
    If wkbSource.Name <> ThisWorkbook.Name Then
```

```
        ExportModules wkbSource
        ImportModules wkbSource
    Else
        MsgBox "Can't run this procedure on myself"
    End If
End Sub

Public Sub ExportModules(wkbSource As Workbook)
    Dim bExport As Boolean
    '    Dim wkbSource As Excel.Workbook
    ''change / made wkbSource an arguement
    '    Dim szSourceWorkbook As String
    ''change / made wkbSource an arguement
    Dim szExportPath As String
    Dim szFileName As String
    Dim cmpComponent As VBIDE.VBComponent
    ''' The code modules will be exported in a folder named.
    ''' VBAProjectFiles in the Documents folder.
    ''' The code below create this folder if it not exist
    ''' or delete all files in the folder if it exist.
    If FolderWithVBAProjectFiles = "Error" Then
        MsgBox "Export Folder not exist"
        Exit Sub
    End If
    On Error Resume Next
    Kill FolderWithVBAProjectFiles & "\*.*"
    On Error GoTo 0
    ''' NOTE: This workbook must be open in Excel.
    '    szSourceWorkbook = ActiveWorkbook.name
    ''change / made wkbSource an arguement
    '    Set wkbSource = Application.Workbooks(szSourceWorkbook)
    '    If wkbSource.VBProject.Protection = 1 Th
    'en                  'change / will check from caller
    '        MsgBox "The VBA in this workbook is protected," & _
    '               "not possible to export the code"
    '        Exit Sub
    '    End If
    szExportPath = FolderWithVBAProjectFiles & "\"
    For Each cmpComponent In wkbSource.VBProject.VBComponents
        bExport = True
        szFileName = cmpComponent.Name
        ''' Concatenate the correct filename for export.
        Select Case cmpComponent.Type
            Case vbext_ct_ClassModule
                szFileName = szFileName & ".cls"
            Case vbext_ct_MSForm
                szFileName = szFileName & ".frm"
            Case vbext_ct_StdModule
                szFileName = szFileName & ".bas"
            Case vbext_ct_Document
                ''' This is a worksheet or workbook object.
                ''' Don't try to export.
```

```
                bExport = False
            End Select
            If bExport Then
                ''' Export the component to a text file.
                cmpComponent.Export szExportPath & szFileName
                ''' remove it from the project if you want
                '''wkbSource.VBProject.VBComponents.Remove cmpComponent
            End If
        Next cmpComponent
        '    MsgBox "Export is ready"
    End Sub

    Public Sub ImportModules(wkbTarget As Workbook)
        'WARNING!
        'DELETES OLD MODULES AND USERFORMS BEFORE IMPORTING NEW
        '    Dim wkbTarget As Excel.Workbook
        ''change / made wkbTarget an arguement
        Dim objFSO As Scripting.FileSystemObject
        Dim objFile As Scripting.File
        '    Dim szTargetWorkbook As String
        ''change / made wkbTarget an arguement
        Dim szImportPath As String
        Dim szFileName As String
        Dim cmpComponents As VBIDE.VBComponents
        If wkbTarget.Name = ThisWorkbook.Name Then
            MsgBox "Select another destination workbook" & _
            "Not possible to import in this workbook "
            Exit Sub
        End If
        'Get the path to the folder with modules
        If FolderWithVBAProjectFiles = "Error" Then
            MsgBox "Import Folder not exist"
            Exit Sub
        End If
        ''' NOTE: This workbook must be open in Excel.
        '    szTargetWorkbook = ActiveWorkbook.name
        ''change / will check from caller
        '    Set wkbTarget = Application.Workbooks(szTargetWorkbook)
        '    If wkbTarget.VBProject.Protection = 1 Then
        '        MsgBox "The VBA in this workbook is protected," & _
        '               "not possible to Import the code"
        '        Exit Sub
        '    End If
        ''' NOTE: Path where the code modules are located.
        szImportPath = FolderWithVBAProjectFiles & "\"
        Set objFSO = New Scripting.FileSystemObject
        If objFSO.GetFolder(szImportPath).Files.Count = 0 Then
            MsgBox "There are no files to import"
            Exit Sub
        End If
        'Delete all modules/Userforms from the ActiveWorkbook
        Call DeleteVBAModulesAndUserForms(wkbTarget)
```

```
    Set cmpComponents = wkbTarget.VBProject.VBComponents
    ''' Import all the code modules in the specified path
    ''' to the ActiveWorkbook.
    For Each objFile In objFSO.GetFolder(szImportPath).Files
        If (objFSO.GetExtensionName(objFile.Name) = "cls") Or _
        (objFSO.GetExtensionName(objFile.Name) = "frm") Or _
        (objFSO.GetExtensionName(objFile.Name) = "bas") Then
            cmpComponents.Import objFile.Path
        End If
    Next objFile
End Sub

Function FolderWithVBAProjectFiles() As String
    Dim WshShell As Object
    Dim fso As Object
    Dim SpecialPath As String
    Set WshShell = CreateObject("WScript.Shell")
    Set fso = CreateObject("scripting.filesystemobject")
    SpecialPath = WshShell.SpecialFolders("MyDocuments")
    If Right(SpecialPath, 1) <> "\" Then
        SpecialPath = SpecialPath & "\"
    End If
    If fso.FolderExists(SpecialPath & "VBAProjectFiles") = False Then
        On Error Resume Next
        MkDir SpecialPath & "VBAProjectFiles"
        On Error GoTo 0
    End If
    If fso.FolderExists(SpecialPath & "VBAProjectFiles") = True Then
        FolderWithVBAProjectFiles = SpecialPath & "VBAProjectFiles"
    Else
        FolderWithVBAProjectFiles = "Error"
    End If
End Function

Function DeleteVBAModulesAndUserForms(wkbSource As Workbook)
    Dim vbproj As VBIDE.VBProject
    Dim vbComp As VBIDE.VBComponent
    Set vbproj = wkbSource.VBProject
    For Each vbComp In vbproj.VBComponents
        '    Debug.Print VBComp.Name
        If vbComp.Type = vbext_ct_Document Then
            'Thisworkbook or worksheet module
            'We do nothing
        Else
            vbproj.VBComponents.Remove vbComp
        End If
    Next vbComp
End Function

''''''''''''''''''''''''''''''''''''''''
'Ron De Bruin Export - Import Modules END'
''''''''''''''''''''''''''''''''''''''''
```

```
Function PickExcelFile() As String
    Dim strFile As String
    Dim fd As Office.FileDialog
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    With fd
        .Filters.Clear
        .Filters.Add "Excel Files", "*.xl*", 1
        .Title = "Choose an Excel file"
        .AllowMultiSelect = False
        .InitialFileName = Environ("USERprofile") & "\Desktop\"
        If .Show = True Then
            strFile = .SelectedItems(1)
            PickExcelFile = strFile
        End If
    End With
End Function

Private Sub chBackup_Click()
    ThisWorkbook.Sheets("ProjectManagerSettings").Range("B5").Value =  _
    chBackup.Value
End Sub

Private Sub chExportProcedures_Click()
    ThisWorkbook.Sheets("ProjectManagerSettings").Range("B6").Value =  _
    chExportProcedures.Value
End Sub

Private Sub chExportXML_Click()
    ThisWorkbook.Sheets("ProjectManagerSettings").Range("B7").Value =  _
    chExportXML.Value
End Sub

Private Sub chFormsProperties_Click()
    ThisWorkbook.Sheets("ProjectManagerSettings").Range("B8").Value =  _
    chPrintCode.Value
End Sub

Private Sub chPrintCode_Click()
    ThisWorkbook.Sheets("ProjectManagerSettings").Range("B4").Value =  _
    chPrintCode.Value
End Sub

Private Sub goToFolder_MouseDown(ByVal Button As Integer, ByVal Shift As  _
Integer, ByVal X As Single, ByVal Y As Single)
    FollowLink Environ("USERprofile") & "\Documents\vbArc\"
End Sub

Private Sub iAdd_MouseDown(ByVal Button As Integer, ByVal Shift As  _
Integer, ByVal X As Single, ByVal Y As Single)
    oAdd.Value = True
    optionsBlank
    iAdd.BorderStyle = fmBorderStyleSingle
```

```
        ExportOptionsHide
    End Sub

    Private Sub iExport_MouseDown(ByVal Button As Integer, ByVal Shift As  _
    Integer, ByVal X As Single, ByVal Y As Single)
        oExport.Value = True
        optionsBlank
        iExport.BorderStyle = fmBorderStyleSingle
        ExportOptionsShow
    End Sub

    Private Sub iImport_MouseDown(ByVal Button As Integer, ByVal Shift As  _
    Integer, ByVal X As Single, ByVal Y As Single)
        oImport.Value = True
        optionsBlank
        iImport.BorderStyle = fmBorderStyleSingle
        ExportOptionsHide
    End Sub

    Private Sub iRename_MouseDown(ByVal Button As Integer, ByVal Shift As  _
    Integer, ByVal X As Single, ByVal Y As Single)
        oRename.Value = True
        optionsBlank
        iRename.BorderStyle = fmBorderStyleSingle
        ExportOptionsHide
    End Sub

    Private Sub iRemove_MouseDown(ByVal Button As Integer, ByVal Shift As  _
    Integer, ByVal X As Single, ByVal Y As Single)
        oDelete.Value = True
        optionsBlank
        iRemove.BorderStyle = fmBorderStyleSingle
    End Sub

    Private Sub iRefresh_MouseDown(ByVal Button As Integer, ByVal Shift As  _
    Integer, ByVal X As Single, ByVal Y As Single)
        oRefresh.Value = True
        optionsBlank
        iRefresh.BorderStyle = fmBorderStyleSingle
        ExportOptionsHide
    End Sub

    Sub optionsBlank()
        iAdd.BorderStyle = fmBorderStyleNone
        iExport.BorderStyle = fmBorderStyleNone
        iImport.BorderStyle = fmBorderStyleNone
        iRename.BorderStyle = fmBorderStyleNone
        iRefresh.BorderStyle = fmBorderStyleNone
        iRemove.BorderStyle = fmBorderStyleNone
    End Sub

    Sub ExportOptionsHide()
```

```
    ExportOptions.Visible = False
    FrameBottom.Top = 36
    Me.Height = 198
End Sub

Sub ExportOptionsShow()
    FrameBottom.Top = 132
    ExportOptions.Visible = True
    Me.Height = 295
End Sub

Private Sub SelectFromList_Click()
    If listOpenBooks.ListIndex = -1 Then
        MsgBox "No book selected"
        Exit Sub
    End If
    Set pmWorkbook = Workbooks(listOpenBooks.List(listOpenBooks.ListIndex) _
    )
    SelectAction
End Sub

Private Sub UserForm_Initialize()
    LoadBooksAndAddins
    LoadCheckboxes
    FormatColourFormatters
End Sub

Sub LoadBooksAndAddins()
    Rem list workbooks
    Dim wb As Workbook
    For Each wb In Workbooks
        If Len(wb.Path) > 0 Then
            If ProtectedVBProject(wb) = False Then listOpenBooks.AddItem  _
            wb.Name
        End If
    Next
    Rem list addins
    Dim vbproj As VBProject
    Dim wbPath As String
    For Each vbproj In Application.VBE.VBProjects
        On Error GoTo ErrorHandler
        wbPath = vbproj.FileName
        If Right(wbPath, 4) = "xlam" Or Right(wbPath, 3) = "xla" Then
            Dim wbName As String
            wbName = Mid(wbPath, InStrRev(wbPath, "\") + 1)
            If ProtectedVBProject(Workbooks(wbName)) = False Then
                If ListboxContains(listOpenBooks, wbName) = False Then  _
                listOpenBooks.AddItem wbName
            End If
        End If
Skip:
    Next vbproj
```

```
        Exit Sub
    ErrorHandler:
        If err.Number = 76 Then GoTo Skip
    End Sub

    Sub LoadCheckboxes()
        Dim ws As Worksheet
        Set ws = ThisWorkbook.Sheets("ProjectManagerSettings")
        chExportSheets.Value = ws.Range("B2").Value
        chExportForms.Value = ws.Range("B3").Value
        chPrintCode.Value = ws.Range("B4").Value
        chBackup.Value = ws.Range("B5").Value
        chExportProcedures.Value = ws.Range("B6").Value
        chExportXML.Value = ws.Range("B7").Value
        chFormsProperties.Value = ws.Range("B8").Value
    End Sub

    Private Sub chExportSheets_Click()
        ThisWorkbook.Sheets("ProjectManagerSettings").Range("B2").Value =  _
        chExportSheets.Value
    End Sub

    Private Sub chExportForms_Click()
        ThisWorkbook.Sheets("ProjectManagerSettings").Range("B3").Value =  _
        chExportForms.Value
    End Sub

    Private Sub ActiveFile_Click()
        Set pmWorkbook = ActiveWorkbook
        SelectAction
    End Sub

    Sub SelectAction()
        If ProtectedVBProject(pmWorkbook) = True Then
            MsgBox "Project of " & pmWorkbook.Name & " is protected."
            Exit Sub
        End If
        Select Case True
            Case oExport.Value = True
                Me.Hide
                ExportProject wb:=pmWorkbook _
                , bMainExport:=chBackup.Value _
                , bSeparateProcedures:=chExportProcedures.Value _
                , exportXML:=chExportXML.Value _
                , PrintCode:=chPrintCode.Value _
                , ExportSheets:=chExportSheets.Value _
                , ExportForms:=chExportForms.Value _
                , ExportFormsProperties:=chFormsProperties.Value
                Me.Show
            Case oImport.Value = True
                ImportComponents pmWorkbook
            Case oRefresh.Value = True
```

```
            RefreshComponents pmWorkbook
        Case oDelete.Value = True
            RemoveComps.Show
        Case oRename.Value = True
            RenameComps.Show
        Case oAdd.Value = True
            AddComps.Show
        Case Else
            '
    End Select
End Sub

Private Sub cInfo_MouseDown(ByVal Button As Integer, ByVal Shift As  _
Integer, ByVal X As Single, ByVal Y As Single)
    uDEV.Show
End Sub

Private Sub Image1_MouseDown(ByVal Button As Integer, ByVal Shift As  _
Integer, ByVal X As Single, ByVal Y As Single)
    ExportSettings.Show
End Sub

Private Sub SelectFile_Click()
    Dim fPath As String
    fPath = PickExcelFile
    If fPath = "" Then Exit Sub
    Set pmWorkbook = Workbooks.Open(FileName:=fPath, UpdateLinks:=0,  _
    ReadOnly:=False)
    SelectAction
    Set pmWorkbook = Nothing
End Sub

Private Sub LBLcolourCode_Click()
    ColorPaletteDialog ThisWorkbook.Sheets("ProjectManagerTXTColour"). _
    Range("GeneralFontBackground"), LBLcolourCode
End Sub

Private Sub LBLcolourComment_Click()
    ColorPaletteDialog ThisWorkbook.Sheets("ProjectManagerTXTColour"). _
    Range("ColourComments"), LBLcolourComment
End Sub

Private Sub LBLcolourKey_Click()
    ColorPaletteDialog ThisWorkbook.Sheets("ProjectManagerTXTColour"). _
    Range("ColourKeywords"), LBLcolourKey
End Sub

Private Sub LBLcolourOdd_Click()
    ColorPaletteDialog ThisWorkbook.Sheets("ProjectManagerTXTColour"). _
    Range("OddLine"), LBLcolourOdd
End Sub
```

--- RemoveComps ---

```
Private Sub Image2_MouseDown(ByVal Button As Integer, ByVal Shift As  _
Integer, ByVal X As Single, ByVal Y As Single)
    uDEV.Show
End Sub

Private Sub Remover_Click()
    If LComponents.ListCount = 0 Then Exit Sub
    If pmWorkbook Is Nothing Then Set pmWorkbook = ActiveWorkbook
    Dim i As Long
    For i = 0 To LComponents.ListCount - 1
        If LComponents.Selected(i) Then
            If oCode.Value = True Then
                ClearComponent pmWorkbook.VBProject. _
                VBComponents(LComponents.List(i, 1))
            ElseIf oComps.Value = True Then
                DeleteComponent pmWorkbook.VBProject. _
                VBComponents(LComponents.List(i, 1))
            End If
        End If
    Next i
    addCompsList
End Sub

Private Sub UserForm_Initialize()
    If pmWorkbook Is Nothing Then Set pmWorkbook = ActiveWorkbook
    addCompsList
    Me.Caption = "Comps of " & pmWorkbook.Name
End Sub

Sub addCompsList()
    LComponents.Clear
    Dim vbComp As VBComponent
    For Each vbComp In pmWorkbook.VBProject.VBComponents
        If vbComp.Name <> "ThisWorkbook" Then
            LComponents.AddItem
            LComponents.List(LComponents.ListCount - 1, 0) =  _
            ComponentTypeToString(vbComp.Type)
            LComponents.List(LComponents.ListCount - 1, 1) = vbComp.Name
            If vbComp.Type = vbext_ct_Document Then
                LComponents.List(LComponents.ListCount - 1, 2) =  _
                GetSheetByCodeName(pmWorkbook, vbComp.Name).Name
            End If
        End If
    Next
    SortListboxOnColumn LComponents, 0
    ResizeControlColumns LComponents
    ResizeUserformToFitControls Me
    Me.Repaint
End Sub
```

--- AddComps ---

```
Private Sub CommandButton1_Click()
    If pmWorkbook Is Nothing Then Set pmWorkbook = ActiveWorkbook
    Dim coll As Collection
    Set coll = New Collection
    Dim element As Variant
    coll.Add Split(Me.tModule.Text, vbNewLine)
    coll.Add Split(Me.tClass.Text, vbNewLine)
    coll.Add Split(Me.tUserform.Text, vbNewLine)
    coll.Add Split(Me.tDocument.Text, vbNewLine)
    Dim typeCounter As Long
    For Each element In coll
        If UBound(element) <> -1 Then
            typeCounter = typeCounter + 1
            AddComponent pmWorkbook, typeCounter, element
        End If
    Next element
    MsgBox typeCounter & " components added to " & pmWorkbook.Name
End Sub

Function AddComponent(wb As Workbook, Module_Class_Form_Sheet As Long,  _
componentArray As Variant)
    Dim compType As Long
    compType = Module_Class_Form_Sheet
    Dim vbproj As VBProject
    Set vbproj = wb.VBProject
    Dim vbComp As VBComponent
    Dim NewSheet As Worksheet
    Dim i As Long
    Dim counter As Long
    On Error GoTo ErrorHandler
    For i = LBound(componentArray) To UBound(componentArray)
        If componentArray(i) <> vbNullString Then
            Select Case compType
                Case Is = 1, 2, 3
                    If ModuleExists(CStr(componentArray(i))) = False Then
                        If compType = 1 Then Set vbComp = vbproj. _
                        VBComponents.Add(vbext_ct_StdModule)         _
                        'module
                        If compType = 2 Then Set vbComp = vbproj. _
                        VBComponents.Add(vbext_ct_ClassModule)         _
                        'class module
                        If compType = 3 Then Set vbComp = vbproj. _
                        VBComponents.Add(vbext_ct_MSForm)        'userform
                    End If
                     vbComp.Name = componentArray(i)
                Case Is = 4
                    If compType = 4 Then
                        Set NewSheet = SheetAdd(CStr(componentArray(i)),  _
                        wb)        'wb.Sheets.Add        'worksheet
                        NewSheet.Name = componentArray(i)
                    End If
```

```
            End Select
        End If
    loop1:
    Next i
        On Error GoTo 0
        Exit Function
    ErrorHandler:
        counter = counter + 1
        componentArray(i) = componentArray(i) & counter
        GoTo loop1
    End Function

    Private Sub UserForm_Click()

    End Sub
```

--- uDEV ---

```
Private Sub LFaceBook_Click()
    FollowLink ("https://www.facebook.com/VBA-Code-Archive- _
    110295994460212")
End Sub

Private Sub LGitHub_Click()
    FollowLink ("https://github.com/alexofrhodes")
End Sub

Private Sub LYouTube_Click()
    FollowLink ("https://bit.ly/2QT4wFe")
End Sub

Private Sub LBuyMeACoffee_Click()
    FollowLink ("http://paypal.me/alexofrhodes")
End Sub

Private Sub LEmail_Click()
    If OutlookCheck = True Then
        MailDev
    Else
        Dim out As String
        out = "anastasioualex@gmail.com"
        CLIP out
        MsgBox ("Outlook not found" & Chr(10) & _
        "DEV's email address" & vbNewLine & out & vbNewLine & "copied to  _
        clipboard")
    End If
End Sub

Sub MailDev()
    'For Tips see: http://www.rondebruin.nl/win/winmail/Outlook/tips.htm
    'Working in Office 2000-2016
    Dim OutApp As Object
    Dim OutMail As Object
    Dim strBody As String
    Set OutApp = CreateObject("Outlook.Application")
    Set OutMail = OutApp.CreateItem(0)
    '    strbody = "Hi there" & vbNewLine & vbNewLine & _
    "This is line 1" & vbNewLine & _
    "This is line 2" & vbNewLine & _
    "This is line 3" & vbNewLine & _
    "This is line 4"
    On Error Resume Next
    With OutMail
        .To = "anastasioualex@gmail.com"
        .CC = vbNullString
        .BCC = vbNullString
        .Subject = "DEV REQUEST OR FEEDBACK FOR -CODE ARCHIVE-"
        .body = strBody
        'You can add a file like this
```

```
            '.Attachments.Add ("C:\test.txt")
            '.Send
            .Display
        End With
        On Error GoTo 0
        Set OutMail = Nothing
        Set OutApp = Nothing
    End Sub
```