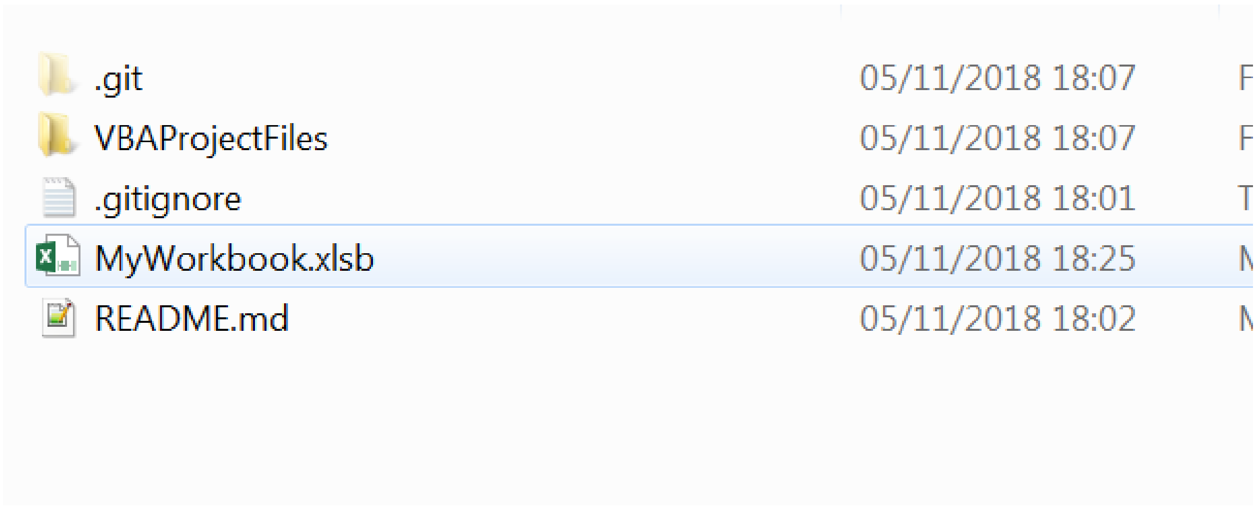
VBA Misc - Usage Instructions

Setup & Import

To set up the VBA misc project on your local, follow these stepa:

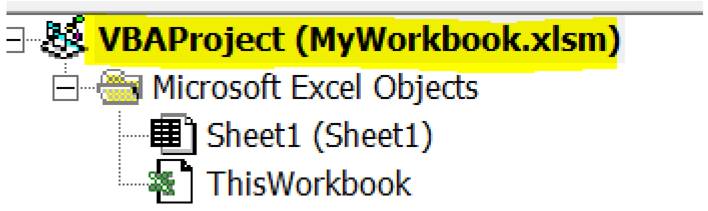
1. Clone the following Git repo anywhere on your machine: https://github.com/ColmBhandal/VbaMisc/tree/master/VBAProjectFiles.
2. Create an Excel workbook of type xlsm or xlsb and drag it into the folder where you cloned the repo.
   a. Alternatively you can copy/move an existing macro-enabled workbook to the repo's folder

> The workbook won't be going into the repo. Rather, consider this workbook as your IDE- in it you will import, modify & run the VBA code. You will also export your code from here. As of 05 Nov 2018, the way the import/export utility works, the workbook must be in the top level folder of the Git repo.



.git	05/11/2018 18:07	F
VBAProjectFiles	05/11/2018 18:07	F
.gitignore	05/11/2018 18:01	T
MyWorkbook.xlsb	05/11/2018 18:25	M
README.md	05/11/2018 18:02	M

3. Open your workbook
4. Hit F11 to open the VBA editor
5. In the Project Pane, right click on the project and then choose Import File

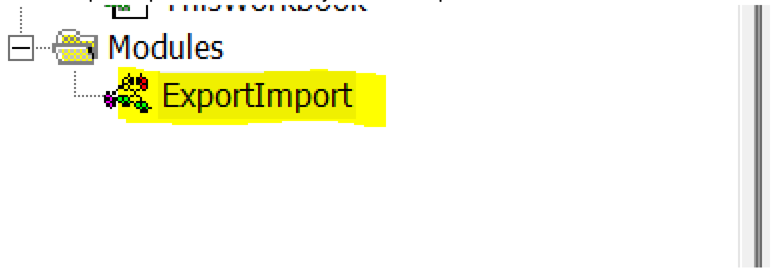


```
VBAProject (MyWorkbook.xlsm)
  Microsoft Excel Objects
    Sheet1 (Sheet1)
    ThisWorkbook
```

6. Navigate to your VBAProjectFiles directory and choose the file called ExportImport.bas

> What we're doing here is bootstrapping the import process - we just do this manual import once, and from then on the imported module will take care of all future imports/exports, including itself.

7. Open the ExportImport Module that's just been imported

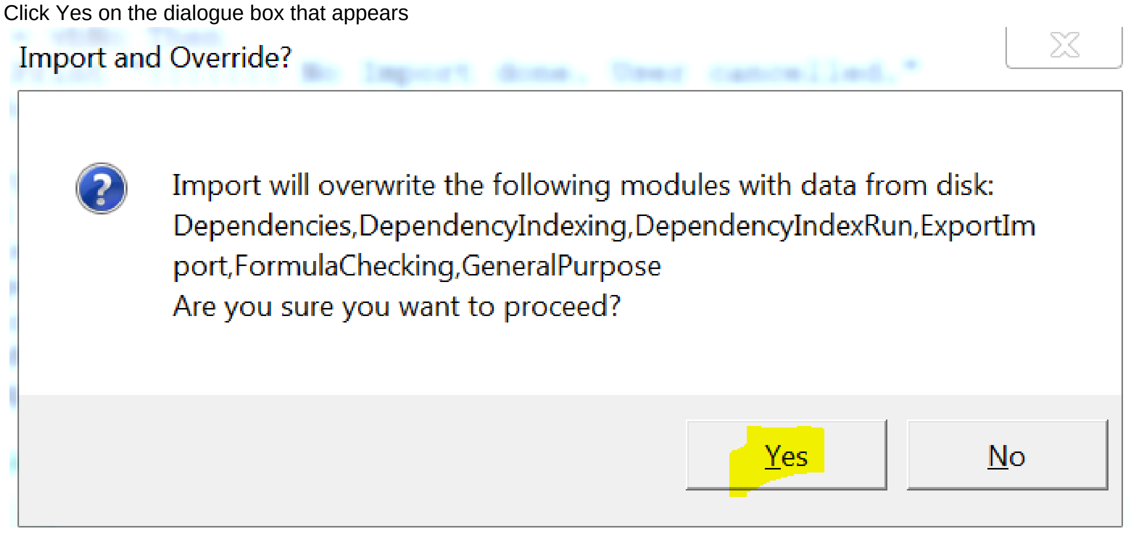


```
Modules
  ExportImport
```

8. Add the following references to your project via Tools References:
    a. Microsoft Visual Basic for Applications Extensibility
    b. Microsoft Scripting Runtime

> See Troubleshooting for more info on how to do this.

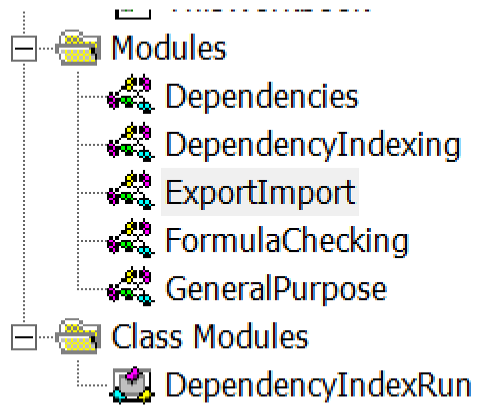
9. Now scroll to the subroutine called ImportModules and hit F5 to run it
10. Click Yes on the dialogue box that appears


Import and Override?

Import will overwrite the following modules with data from disk:
Dependencies,DependencyIndexing,DependencyIndexRun,ExportImport,FormulaChecking,GeneralPurpose
Are you sure you want to proceed?

> This is just a warning to the user that their import will overwrite any existing modules with the same name as the ones being imported.

That's it! You should now have all modules imported:


Modules
- Dependencies
- DependencyIndexing
- ExportImport
- FormulaChecking
- GeneralPurpose

Class Modules
- DependencyIndexRun

Export

Once you've imported the modules, you can start working on them like any other coding project. When you're ready to commit, you "save" your modules by running the export subroutine. To do so, scroll to the subroutine ExportModules and hit F5. This overwrites the entire directory of exported files and replaces it with the modified files.

White Listing & New Files

This project uses whitelisting to control which files go into the repo. This means you can have other VBA code in your Excel file that doesn't get exported/imported, which is a must have. For this reason, you have to explicitly register any new modules with the ExportImport module in order for them to be picked up by export/import. To do so, simply add the new module name, ***without*** the file extension, to the comma-separated string of names in the whiteList function in the ExportImport module:

White List of Modules to Work With

```
'Only modules on this list will get imported/exported
'Add your modules to the whiteList variable, separated by commas
Private Function whiteList() As String
    whiteList =
"Dependencies,DependencyIndexing,DependencyIndexRun,ExportImport,FormulaCh
ecking,GeneralPurpose"
End Function
```

That's it for export!

Git Workflow

With the import/export utilities above, Git workflow is almost the same as for any other project e.g. a Java coding project. For example, here's roughly how you'd do feature branch development:

1. Git pull to update your master branch
2. Create your feature branch
3. Import to pull the latest changes into Excel
4. Work on your files in Excel
5. Export your changes to your local folder
6. Commit the changes to your local Git
7. Push your changes

We won't cover merging/conflicts here, as that's a staple of any Git project.

Warning

Do not change line #2 of the ExportImport utility:

White List of Modules to Work With

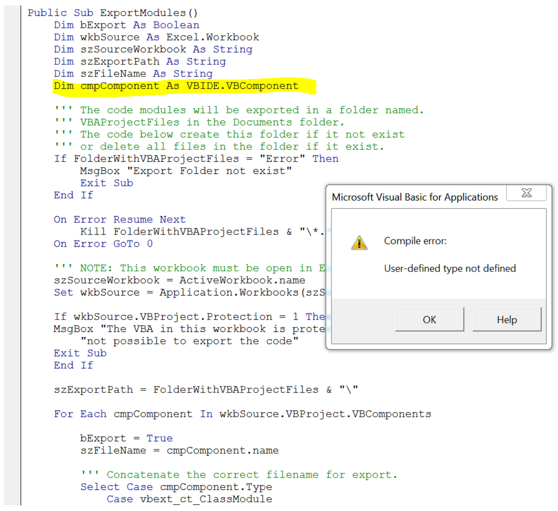
```
Const IOEXP_UNIQUE_STRING = "zn8AiLJcRXREAfOSpY"
```

Without going too deep into the technical details, this line of code allows the ExportImport module to export/import itself. But for this to work the constant must be defined on line #2, not anywhere else!

Troubleshooting

User-defined type not defined.

VBIDE.VBComponent



```
Public Sub ExportModules()
    Dim bExport As Boolean
    Dim wkbSource As Excel.Workbook
    Dim szSourceWorkbook As String
    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 E
    szSourceWorkbook = ActiveWorkbook.name
    Set wkbSource = Application.Workbooks(szS

    If wkbSource.VBProject.Protection = 1 The
    MsgBox "The VBA in this workbook is prote
        "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
```

Microsoft Visual Basic for Applications

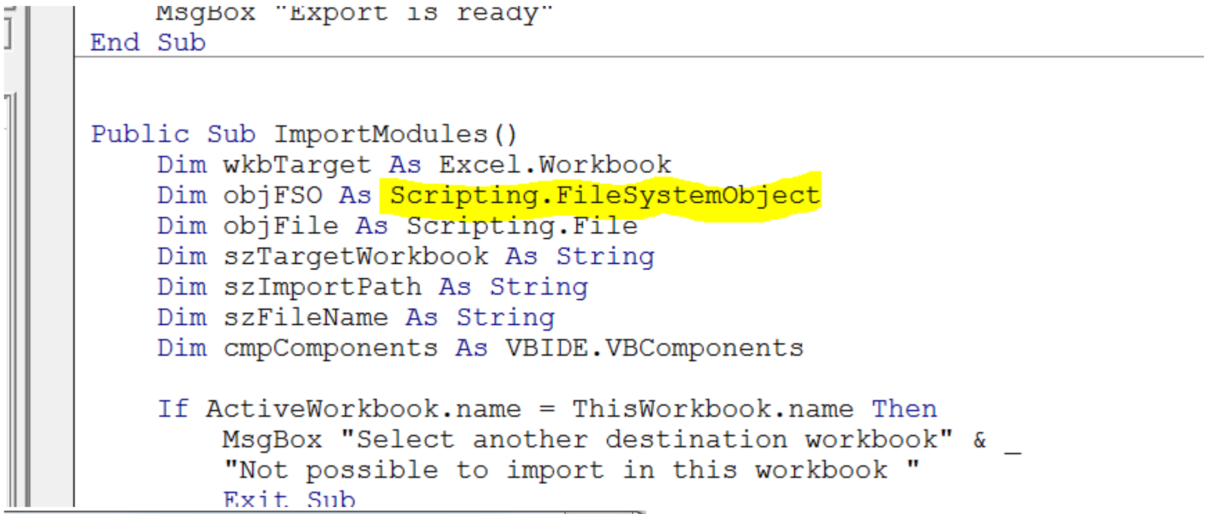
Compile error:

User-defined type not defined

The problem is with VBIDE.VBComponent. The solution given on social.msdn.microsoft.com is to add a reference to the Microsoft Visual Basic for Applications Extensibility library from Tools References.

Scripting.FileSystemObject

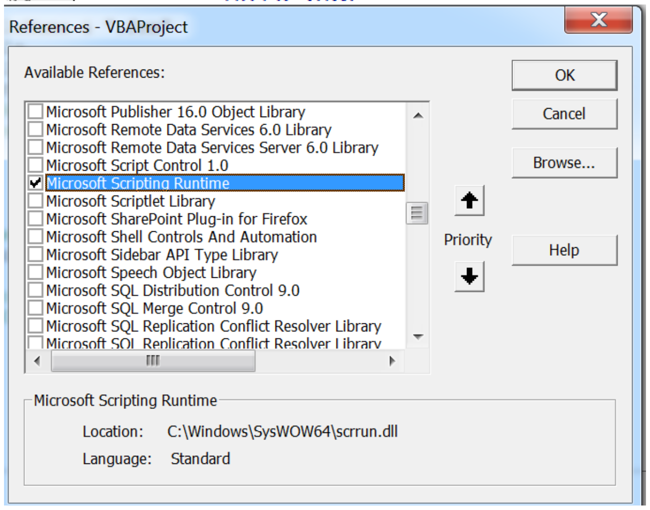
Similar to the last one, except this time we need to import Microsoft Scripting Runtime according to https://stackoverflow.com/questions/3233203/how-do-i-use-filesystemobject-in-vba.



```
    MsgBox "Export is ready"
End Sub

Public Sub ImportModules()
    Dim wkbTarget As Excel.Workbook
    Dim objFSO As Scripting.FileSystemObject
    Dim objFile As Scripting.File
    Dim szTargetWorkbook As String
    Dim szImportPath As String
    Dim szFileName As String
    Dim cmpComponents As VBIDE.VBComponents

    If ActiveWorkbook.name = ThisWorkbook.name Then
        MsgBox "Select another destination workbook" & _
        "Not possible to import in this workbook "
        Exit Sub
```



References - VBAProject

Available References:

- [ ] Microsoft Publisher 16.0 Object Library
- [ ] Microsoft Remote Data Services 6.0 Library
- [ ] Microsoft Remote Data Services Server 6.0 Library
- [ ] Microsoft Script Control 1.0
- [✓] Microsoft Scripting Runtime
- [ ] Microsoft Scriptlet Library
- [ ] Microsoft SharePoint Plug-in for Firefox
- [ ] Microsoft Shell Controls And Automation
- [ ] Microsoft Sidebar API Type Library
- [ ] Microsoft Speech Object Library
- [ ] Microsoft SQL Distribution Control 9.0
- [ ] Microsoft SQL Merge Control 9.0
- [ ] Microsoft SQL Replication Conflict Resolver Library
- [ ] Microsoft SQL Replication Conflict Resolver Library

Microsoft Scripting Runtime

Location: C:\Windows\SysWOW64\scrrun.dll

Language: Standard