FREE eBook

LEARNING outlook-vba

Free unaffiliated eBook created from
Stack Overflow contributors.

#outlook-vba

Table of Contents

About 1

Chapter 1: Getting started with outlook-vba 2

Remarks 2

Examples 2

Introduction 2

Outlook Visual Basic for Applications 3

Advanced topics 3

Chapter 2: Introduction Part 1: Gaining access to Outlook's Visual Basic Editor 4

Introduction 4

Examples 4

1.1 Gaining access to Outlook 2003's Visual Basic Editor 4

1.2 Gaining access to the Visual Basic Editor in Outlook 2007 and later 4

1.3 Getting started with the Visual Basic Editor 8

1.4 What you should remember from this part of the tutorial 13

Chapter 3: Introduction Part 2: Stores and top-level folders 14

Introduction 14

Examples 14

2.1 Expected prior knowledge 14

2.2 Stores 14

2.3 Top level folders 16

2.4 What you should remember from this tutorial 17

Chapter 4: Introduction Part 3: Stores and all their folders 18

Introduction 18

Examples 18

3. 0 Contents 18

3.1 Function GetFldrNames() which is needed for several of the demonstration macros 18

3.2 Referencing a default folder 19

3.3 Referencing any folder within any accessible store 20

3.4 Listing the names of every folder within every accessible store 21

3.5 Moving a folder from one parent folder to another 22

3.6 What you should remember from this part of the tutorial 22

Credits 24

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: outlook-vba

It is an unofficial and free outlook-vba ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official outlook-vba.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with outlook-vba

Remarks

This section provides an overview of what outlook-vba is, and why a developer might want to use it.

It should also mention any large subjects within outlook-vba, and link out to the related topics. Since the Documentation for outlook-vba is new, you may need to create initial versions of those related topics.

Examples

Introduction

There are currently three topics introducing Outlook VBA and at least three more are planned.

Part 1 describes how to get access to the Visual Basic Editor.

If you are a user of Outlook 2003 and a user of Excel VBA, you will learn little for this part since accessing the Outlook Visual Basic Editor is the same as accessing the Excel Visual Basic Editor.

With Outlook 2007 and later, the `Developer` tab. which gives access to the Visual Basic Editor, is not displayed for a new installation. To display the `Developer` tab, you must perform an number of steps which are described in this part. There is no code in this part.

Parts 2 and 3 describe stores and folders which are where Outlook stores data. You could think of them as the equivalent of Excel's workbooks and worksheets. The division between part 2 and 3 is somewhat arbitrary. Part 2 describes stores and folders and includes macros to display the names of all accessible stores and the top level folders within those stores. Part 3 includes macro for accessing lower level folders. One pair of macros uses recursion which a new programmer may find difficult to understand. The reader should aim to understand all the code in Part 2. It would however be legitimate to understand what that pair of macros does but not understand how they achieve their objective.

Part 4, the next part to be written, will introduce `MailItems` which hold emails. Part 3 includes a macro to move a folder from one parent to another but most macros operate on the objects contained within folders not folders themselves. Judging from the questions on Stack overflow, `MailItems` are of most interest to programmers.

Part 5 will introduce `CalendarItems` which hold appointments. Part 6 will introduce the creation of new Excel workbooks from Outlook and the reading and updating of existing workbooks. Part 7 will introduce Events unless some more immediately important topic is identified.

It is important to understand this is an introduction to Outlook VBA not an introduction to VBA. Part 2 gives some guidance on where to get information on VBA but since the language is the same

across all Office products, a description of it belongs outside this introduction to Outlook VBA.

Outlook Visual Basic for Applications

Visual Basic for Applications (VBA) is the macro language behind all Microsoft Office products and is essentially identical across all Office products. What differs from product to product is the Object model. Excel has workbooks, worksheets and cells. Access has tables and attributes. Outlook has folders, emails and appointments. It is the Object Model that makes Excel VBA different from Outlook VBA.

Advanced topics

The various parts of the introduction aim to give the information that any programmer new to Outlook VBA would need. Much of the code was originally developed with Outlook 2003 and has been tested with Outlook 2016. It should work unchanged with any intermediate version.

New functionality has been introduced since Outlook 2003 which programmers will wish/need to access. It is envisaged that "advanced topics" will be written to describe this functionality.

Read Getting started with outlook-vba online: https://riptutorial.com/outlook-vba/topic/8111/getting-started-with-outlook-vba

Chapter 2: Introduction Part 1: Gaining access to Outlook's Visual Basic Editor

Introduction

Gaining access to Outlook's Visual Basic Editor, inserting your first module and renaming that module.

Expected prior knowledge: You are an Outlook user.

With Outlook 2003, you can immediately select the Visual Basic Editor. With later versions, you must add the Developer tab before you can select the Visual Basic Editor.

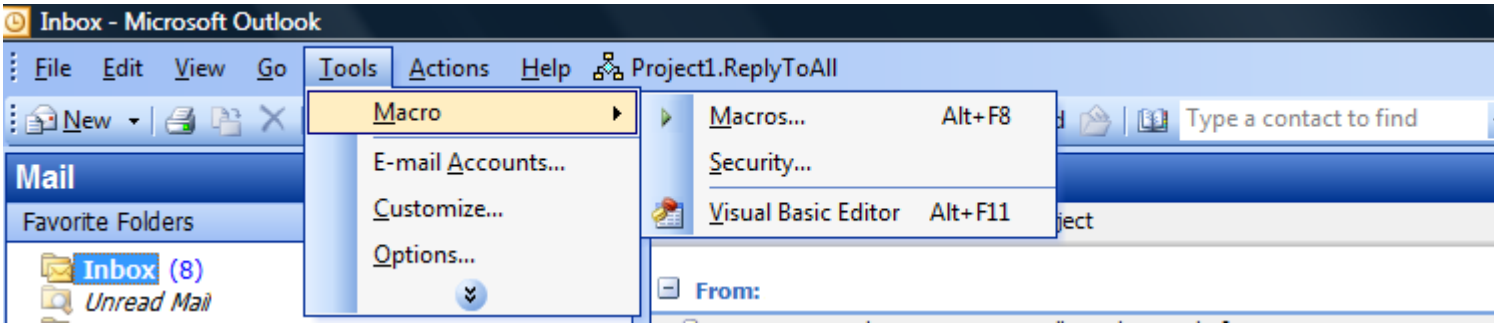
Examples

1.1 Gaining access to Outlook 2003's Visual Basic Editor

All images are from UK versions of Outlook. I know that some names are translated into the local language for other versions and I assume that most of the names for the tabs are translated. Probably the sequence of tabs is unchanged in non-English versions. Alternatively, you will need to look at your tabs and decide which would be equivalent of, for example, “Tools”

With Outlook 2003 open, the top of the window might look like:

Click Tools and move the cursor down to Macros to see:



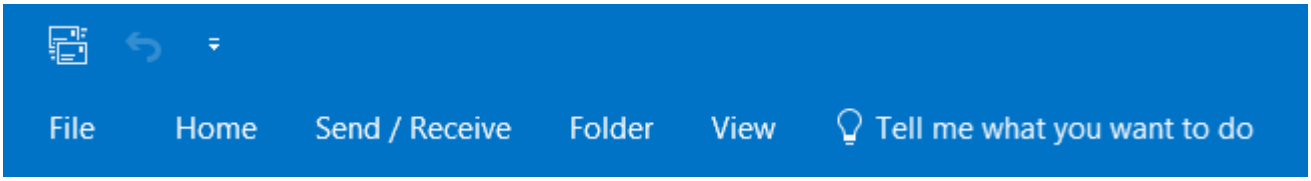
Move the cursor right then down and click `Visual Basic Editor`. Alternatively, exit the selections and click `Alt`+`F11`.

1.2 Gaining access to the Visual Basic Editor in Outlook 2007 and later

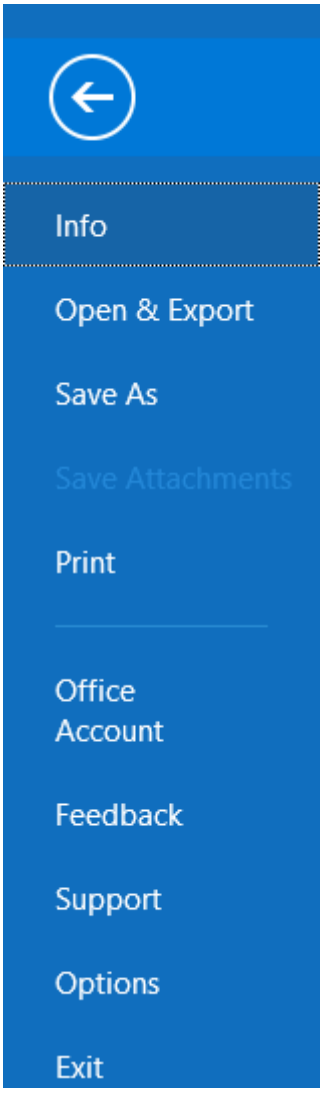
All images in this section are from the UK version of Outlook 2016. I know that some names are translated into the local language for other versions and I assume that most of the names for the tabs are translated. Probably the sequence of tabs is unchanged in non-English versions. Alternatively, you will need to look at your tabs and decide which would be equivalent of, for example, “Tools”

Outlook 2010 windows are formatted differently but are essentially identical. I understand other versions are also essentially identical to Outlook 2016.

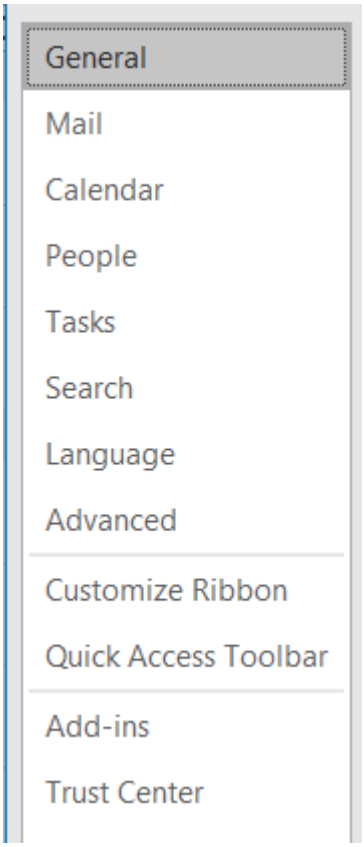
The top of the main window might look like:



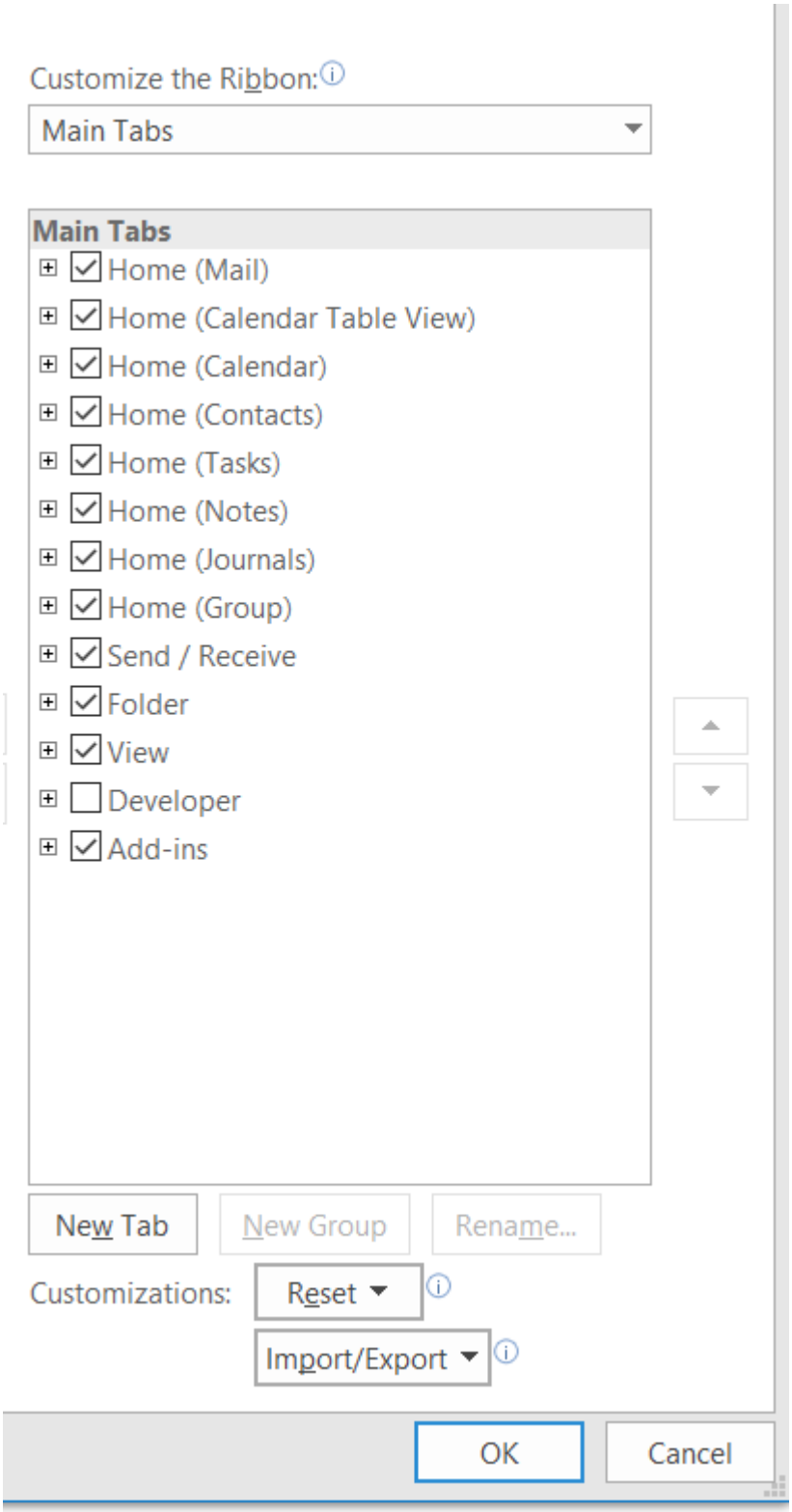
Click `File`, on the left, to get the following on the left of the window:



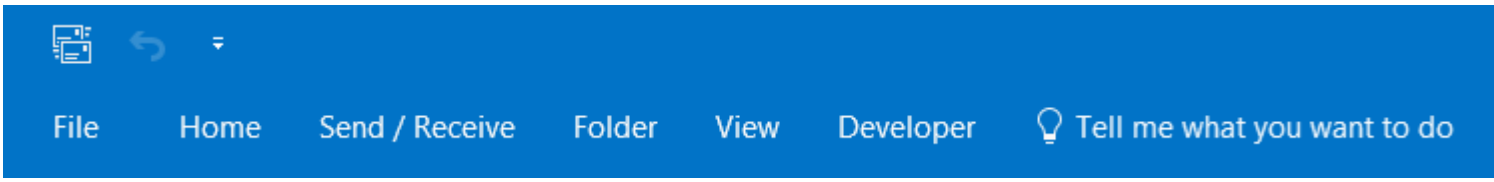
Click `Options`, near the bottom, to get the following on the left of the window:



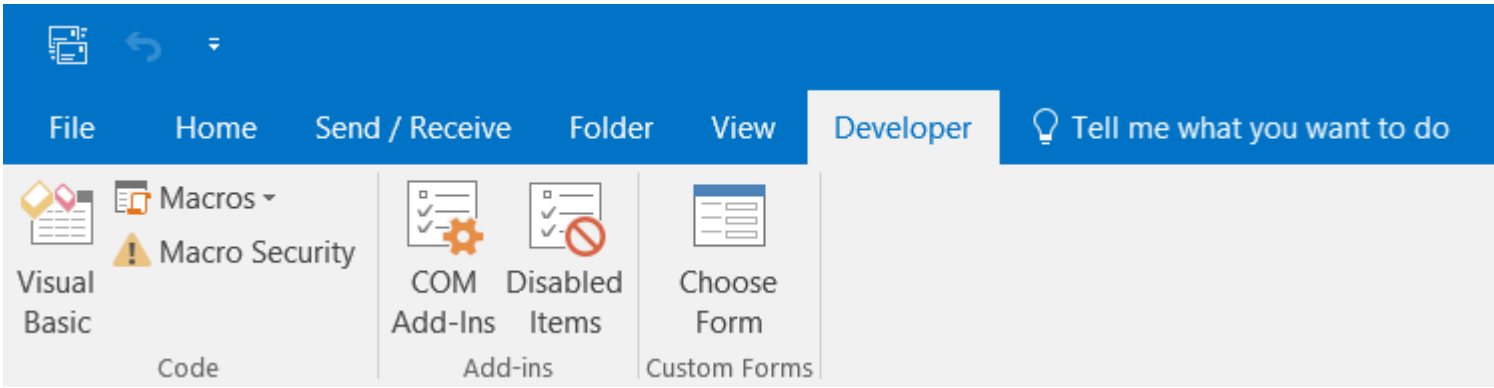
Click `Customize Ribbon`, half way down. to get the following on the right of the window:



Click the box next to “Developer”, near the bottom, to get a tick then click `OK`, at the bottom. The main window will reappear but will have changed to:



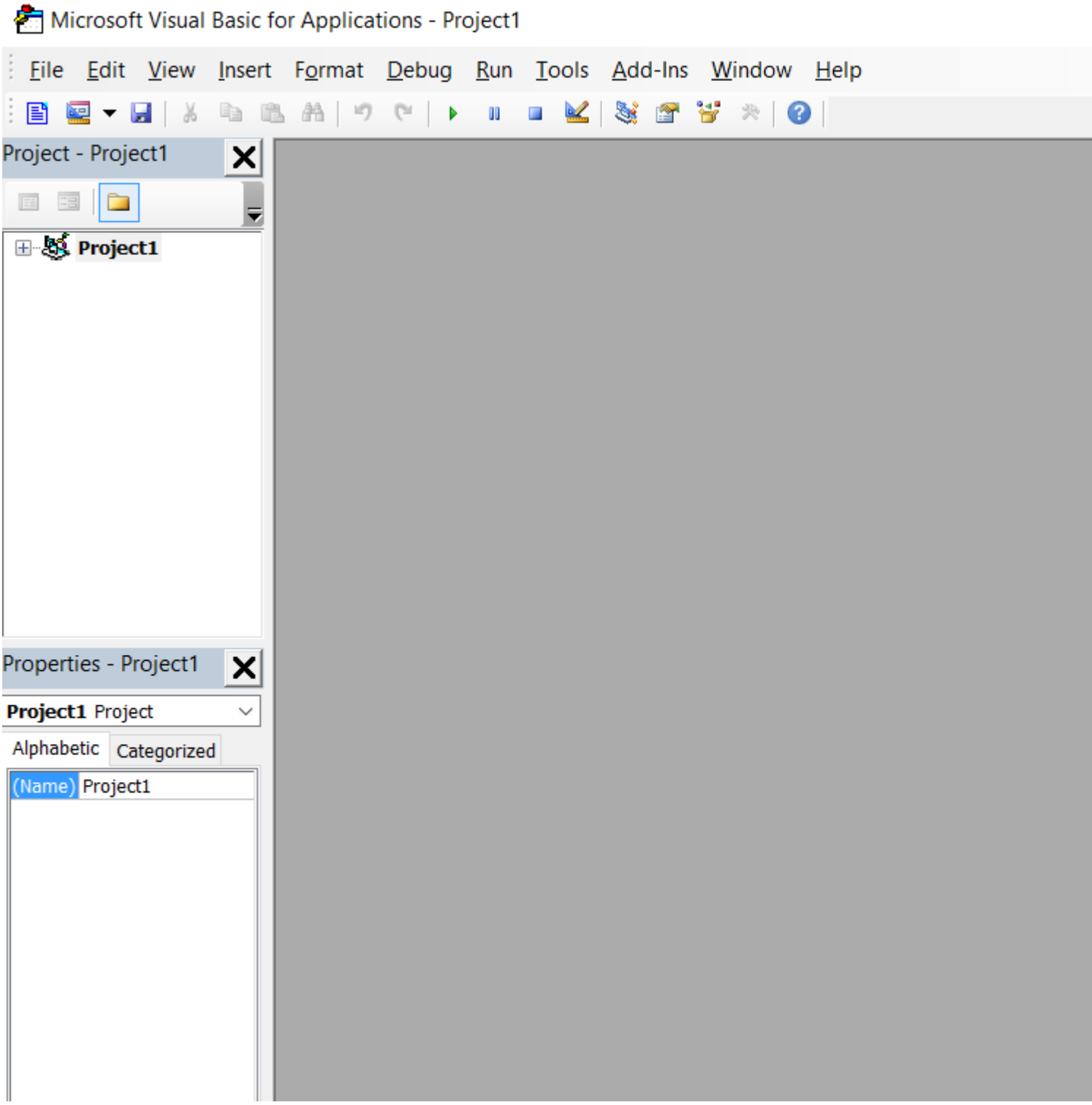
Click the new `Developer` tab to get:



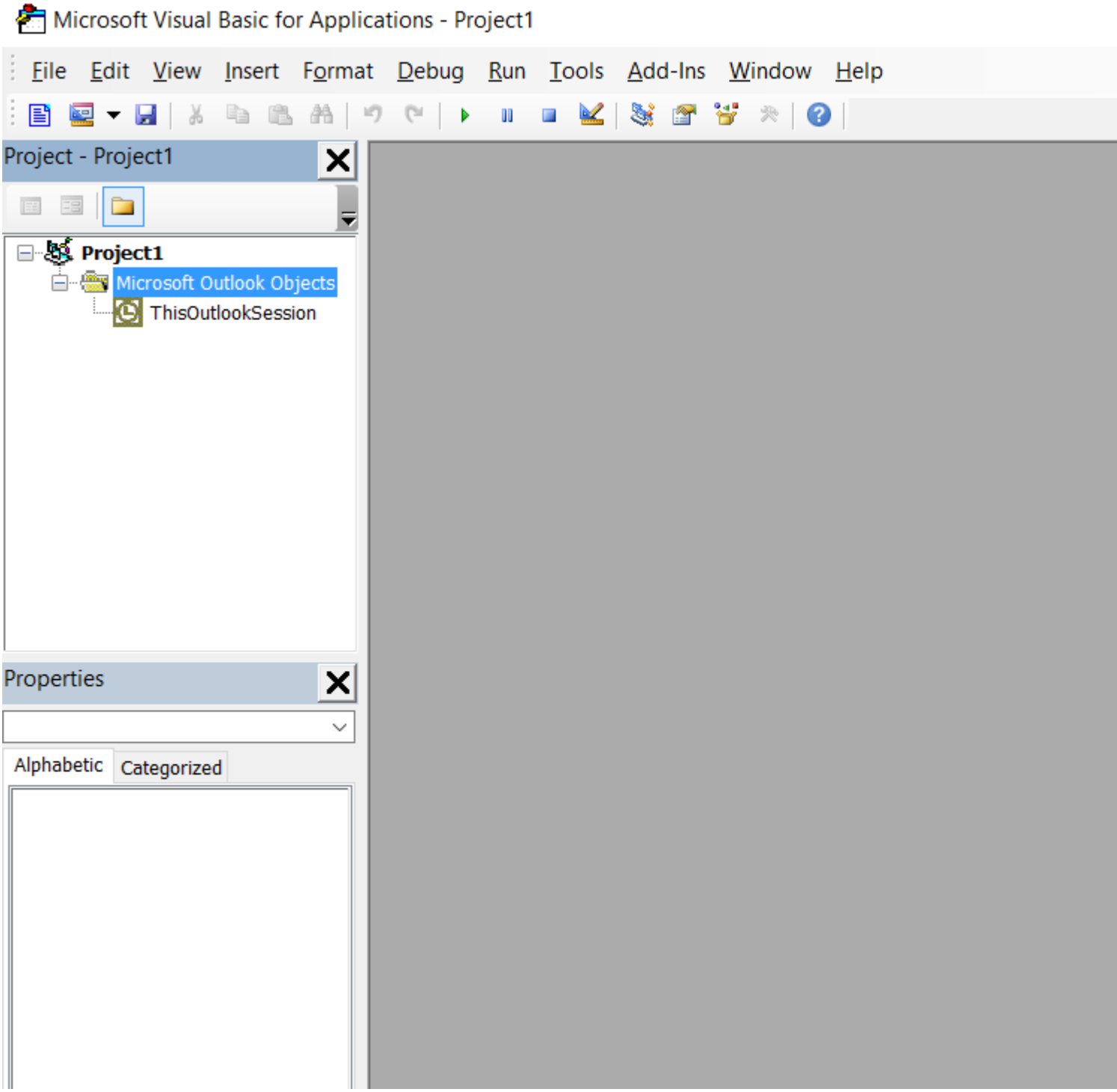
Click `Visual Basic`, on the left, to select the Visual Basic Editor.

1.3 Getting started with the Visual Basic Editor

The images in this section are all from Outlook 2016 but they could have come from Outlook 2003. Outlook VBA may have changed over the years but to my eyes the VBA Editor has not. Whichever version you have you will see something like:

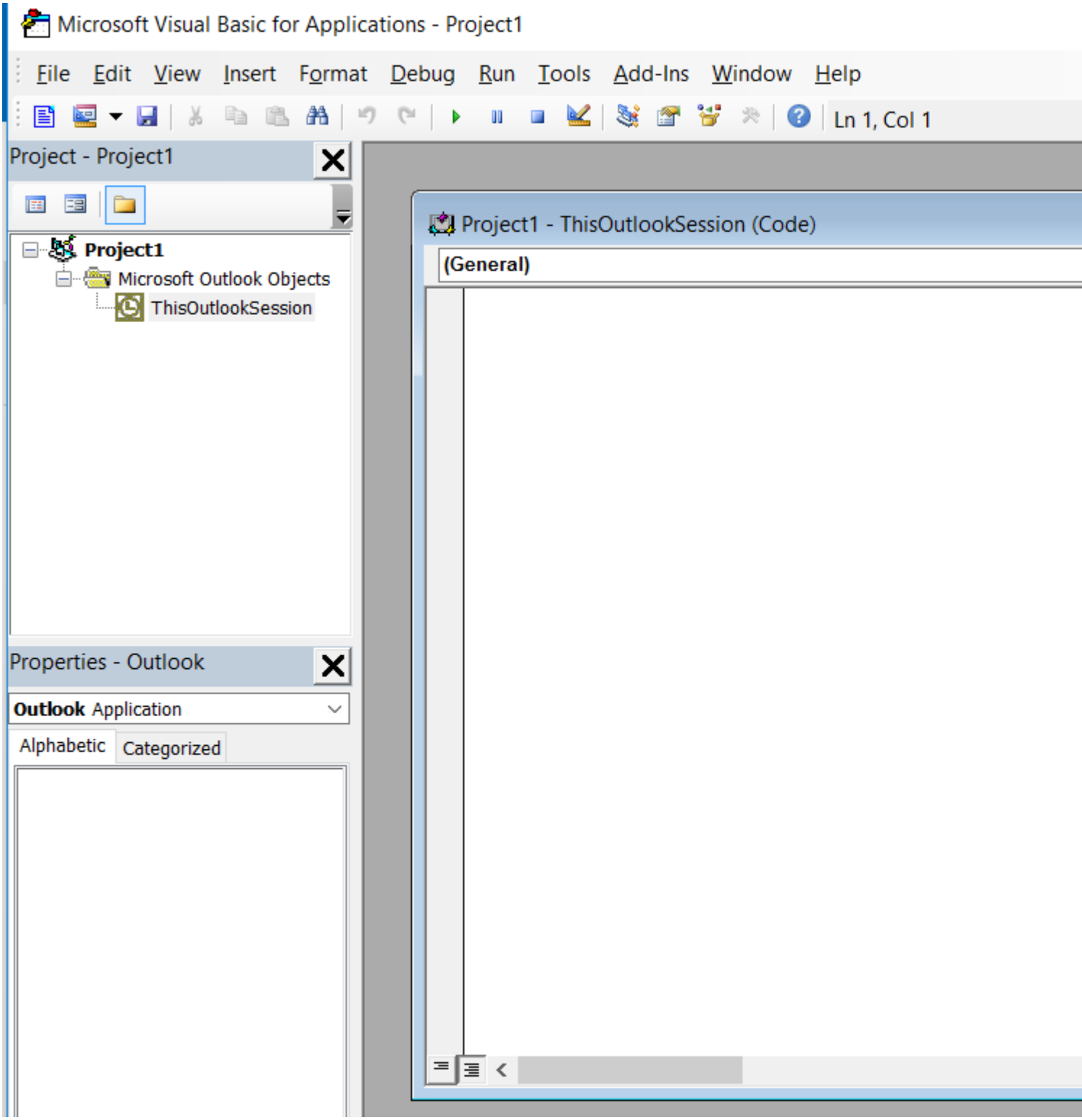


Above there is a “+” against "**Project1**". If you have a “+” click it and then the “+” against "Microsoft Outlook Objects" to get:



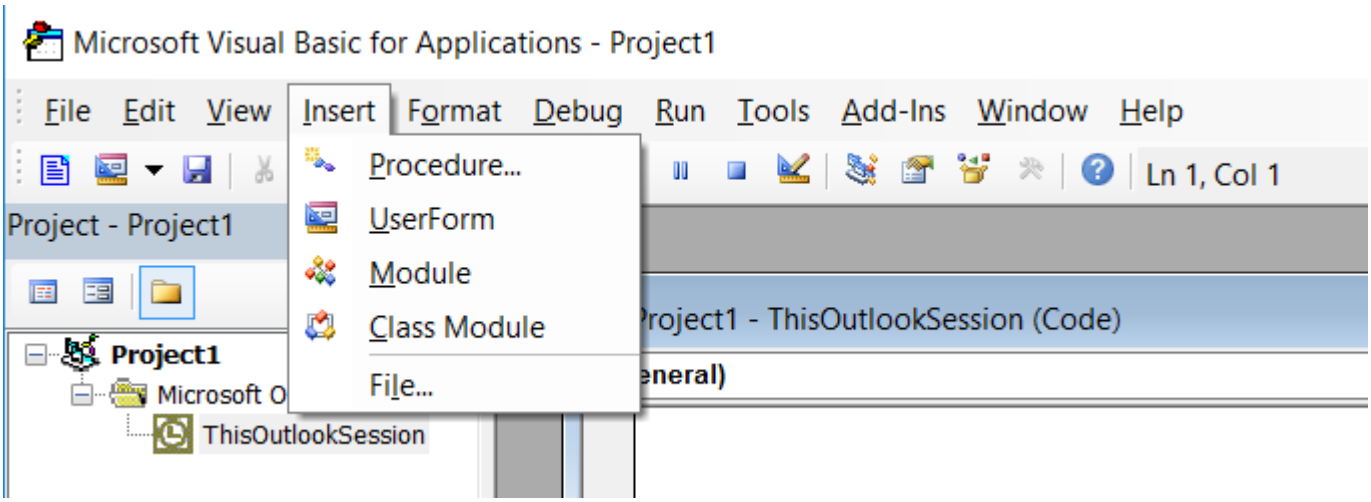
The Properties window may not be present or, if present, may be positioned elsewhere within the VB Editor window. We do not need it for the moment. You can close it by clicked the cross and can use `F4` to make it visible again at any time. I do not normally have it visible because I do not need access to Properties most of the time and my Project Explorer list occupies most of the left side. I suggest you keep it visible until it becomes a nuisance.

If you click `ThisOutlookSession`, either the grey area will turn white or, as in the image below, a code window will appear within the grey area:

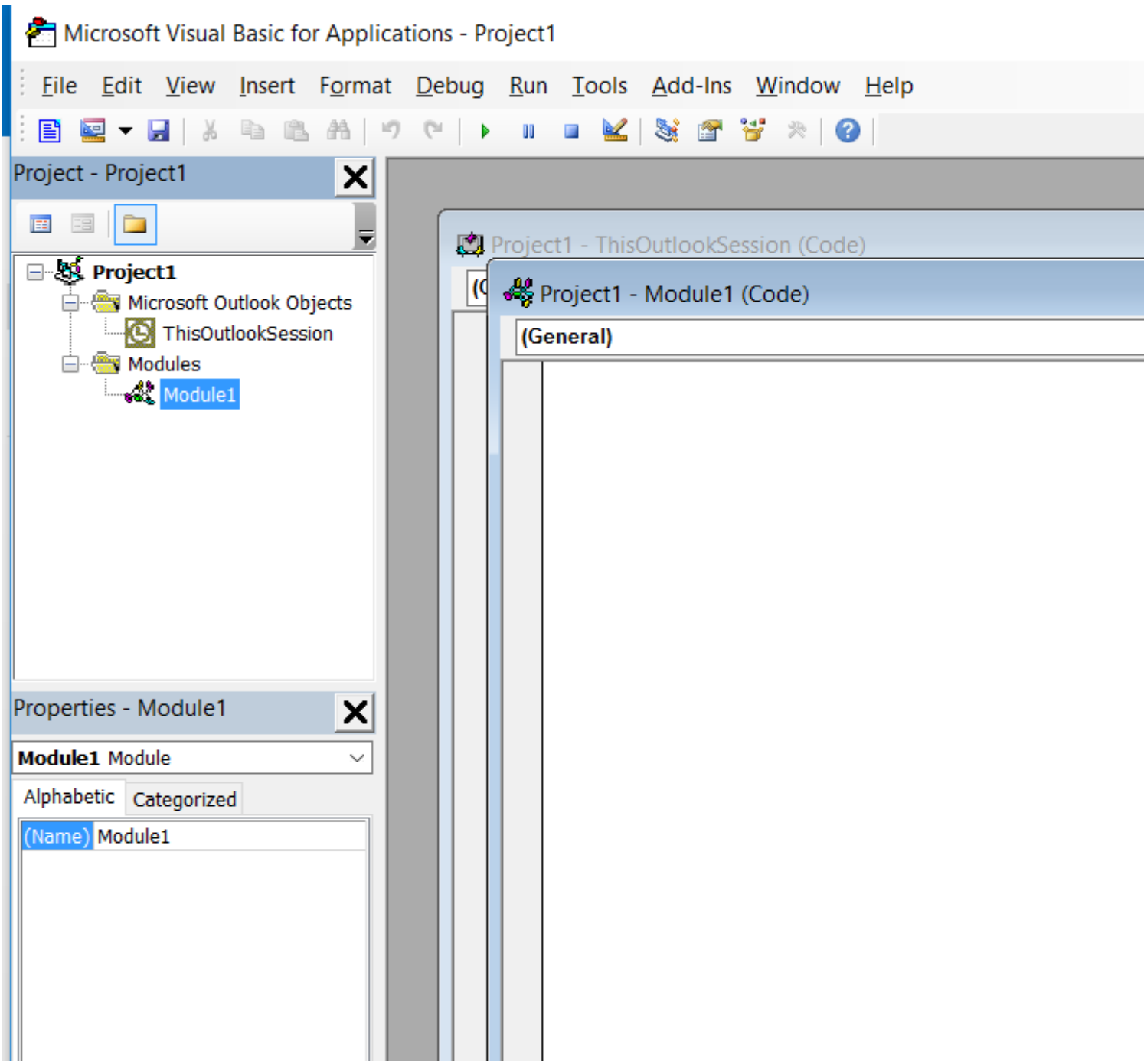


You can type any code into this code window. However, event routines (which are discussed towards the end of this tutorial) must be typed into this code window. I recommend you reserve the ThisOutlookSession code area for event routines.

Instead, click `Insert` to get:



Click on `Module` to add a module:



My new module is named “Module1”. If your version of Outlook is a non-English version, your module will have an equivalent name in your language. You can add more modules which will be named "Module2", "Module3" and so on.

If I am creating an Excel workbook, for which I only need one module, I might leave the name as “Module1”. But with Outlook, all my macros have to go here so I have lots of modules. Over the years I have written many routines which I reuse repeatedly. I have one module for general VBA routines, another for routines to access Excel, another for Outlook VBA routines and then one module per Outlook task I perform with macros. If you look at the Properties window you will see the only property of a module is its name. Click on the “Module1” against “Name” and you can change it to any valid (starts with a letter, contains letters and number only, etc.) name. You get strange errors if a module and a procedure have the same name so I start all my module names with “Mod” and I do not use this prefix for my procedures. Why not rename this module “ModIntro” or similar ready for the next part of this tutorial?

These code areas and like the data entry areas of any editor. Click on the code area to select it and type your code or paste in code copied from elsewhere such as the next section of this tutorial.

1.4 What you should remember from this part of the tutorial

- Did your version of Outlook need you to add the Development tab? If so, you will not need to repeat this process until you next have a new Outlook installation. Come back here when that happens.
- Remember how to enter the Visual Basic Editor.
- Remember how to create and rename a module.

Read Introduction Part 1: Gaining access to Outlook's Visual Basic Editor online: https://riptutorial.com/outlook-vba/topic/8877/introduction-part-1--gaining-access-to-outlook-s-visual-basic-editor

Chapter 3: Introduction Part 2: Stores and top-level folders

Introduction

First part of an introduction to stores and the folders they contain. Contains macros to display (1) the names of accessible stores and (2) the names of accessible stores and the top level folders within them.

Examples

2.1 Expected prior knowledge

- You are an Outlook user and understand terms such as “email”, “received time”, “subject” and “Folder Pane”.
- You know how to access Outlook’s Visual Basic Editor and create a module. See Introduction Part 1 if necessary.
- You have at least a basic knowledge of VBA. I declare Subroutines and variables without explanation. I use Withs, Ifs and Loops without explanation. I tell you something is a collection. I tell you to copy code to a module and run it. There are many online tutorials although most are for Excel VBA and concentrate more on using the language with Excel than on the language. Searching for “VBA tutorial” brings up some that concentrate on the language more than the application that look satisfactory.
- You are not required to know the Outlook Object Model; this tutorial introduces you to a small part of it.

2.2 Stores

Outlook stores emails, calendar items, notes, tasks and so on in files known as **Stores**. If you look at your Folder Pane you will see something like:

```
Aaaaaaaaaa
  Inbox
  Drafts
  Deleted Items
  :  :

Bbbbbbbbbb
  Inbox
  Drafts
  Deleted Items
  :  :

Ccccccccc
:  :
```

"Aaaaaaaaaa", "Bbbbbbbbbb" and "Ccccccccc" are the user or display names of Stores. I have always accepted Outlook defaults for these names which have changed over the years. Once the default was my name now it is my email address. The filename for these stores may be the same but with an extension such as PST or OST or may be something completely different. A VBA macro needs the user name to access a store and is not concerned with the file names or the extension.

You can have as many stores as you wish. I have “Outlook data file” which was created for me when I installed Outlook. When I added accounts for my email addresses, Outlook created new stores named for the email address such as “JohnDoe@hotmail.com” and “DoeJohn@gmail.com”. To reduce the size of my main store I save old emails in stores with names such “Archive 2015”.

If you are a business user, you may have access to shared stores or to the stores of colleagues.

The macros below show three different ways of listing the stores you can access. I suggest you create a new module to hold the code below and to use `F4` to access the module’s properties so you can be name it as “ModIntro” or some other name of your choice. If you completed Part 1 of this series, you will already have such a module.

Copy these macros to a module and test that each gives the same output.

```
Sub ListStores1()

  Dim InxStoreCrnt As Integer
  Dim NS As NameSpace
  Dim StoresColl As Folders

  Set NS = CreateObject("Outlook.Application").GetNamespace("MAPI")
  Set StoresColl = NS.Folders

  For InxStoreCrnt = 1 To StoresColl.Count
    Debug.Print StoresColl(InxStoreCrnt).Name
  Next

End Sub
Sub ListStores2()

 Dim StoresColl As Stores
 Dim StoreCrnt As Store

 Set StoresColl = Session.Stores

 For Each StoreCrnt In StoresColl
   Debug.Print StoreCrnt.DisplayName
 Next

End Sub
Sub ListStores3()

  Dim InxStoreCrnt As Long

  With Application.Session
    For InxStoreCrnt = 1 To .Folders.Count
      Debug.Print .Folders(InxStoreCrnt).Name
    Next
  End With
```

```
End Sub
```

You will find with VBA that there are often several methods of achieving the same effect. Above I have shown three methods of accessing the stores. You do not need to remember them all – pick your own favourite – but you do need to be aware that there are several methods because other people, whose code you may need to study, will have different favourites.

The variables `StoresColl` in macros `ListStores1()` and `ListStores2()` are both collections but hold different types of object: `Store` and `Folder`. A `Store` object can only reference a file on your disc. A `Folder` can reference a file on disc but can also reference folders within a store such as “Inbox” and “Sent Items”. `Stores`, `Folders`, `Store` and `Folder` are all part of the Outlook Object Model. This tutorial series introduces you to the model but it is not a formal definition. If you want a formal definition, type “outlook vba object model” into your favourite search engine. Make sure you look at the VBA version of the model.

2.3 Top level folders

In my Folder Pane example above, I only list three standard folders: “Inbox”, “Drafts” and “Deleted Items”. There are other standard folders and you can create as many folders of your own as you wish. Some people create folders under Inbox but I prefer to create new folders at the same level as Inbox. Your folders can have sub-folders which can have their own sub-folders to any depth.

The following macro will produce a listing of the form:

```
A
   A1
   A2
   A3
B
   B1
   B2
C
   C1
   C2
   C3
   C4
```

where A, B and C are stores and A1, B1, C1 and so on are folders within A, B and C. If A1, B1, C1 and so on have sub-folders, they will not be listed by this macro. Accessing more deeply nested folders will be covered in the next part of this tutorial.

```
Sub ListStoresAndTopLevelFolders()

  Dim FldrCrnt As Folder
  Dim InxFldrCrnt As Long
  Dim InxStoreCrnt As Long
  Dim StoreCrnt As Folder

  With Application.Session
    For InxStoreCrnt = 1 To .Folders.Count
      Set StoreCrnt = .Folders(InxStoreCrnt)
```

```
      With StoreCrnt
        Debug.Print .Name
        For InxFldrCrnt = .Folders.Count To 1 Step -1
          Set FldrCrnt = .Folders(InxFldrCrnt)
          With FldrCrnt
            Debug.Print "   " & .Name
          End With
        Next
      End With
    Next
  End With

End Sub
```

2.4 What you should remember from this tutorial

- A store is a file in which Outlook stores emails, calendar items, notes, tasks and so on.
- A store may contain Outlook standard folders such as “Inbox” and “Sent Items”.
- A store may also contain user created folders.
- Both Outlook standard folders and user created folders may contain user created sub-folders, sub-sub-folders and so on to any depth.
- How to list stores.
- How to list stores and the top level folders within those stores.

Confession: I do not remember either of the “Hows”. I have subroutines and functions that remember for me.

Read Introduction Part 2: Stores and top-level folders online: https://riptutorial.com/outlook-vba/topic/8876/introduction-part-2--stores-and-top-level-folders

Chapter 4: Introduction Part 3: Stores and all their folders

Introduction

Completes the introduction to stores and folders started in part 2 of this tutorial

Expected prior knowledge: You have studied part 2 of this tutorial or are already familiar with its contents.

Examples

3. 0 Contents

- How to reference any accessible folder.
- How to get the full name of a referenced folder.
- A pair of routines that together will list every folder within every accessible store.
- A routine to move a folder from one parent folder to another.

3.1 Function GetFldrNames() which is needed for several of the demonstration macros

A number of the demonstration macros within this part requires a function which I will explain later. For the moment, please just copy `GetFldrNames()` to a suitable module. I use this function frequently and keep it, and other like it that I use in many different macros, in a module named “ModGlobalOutlook”. You might like to do the same. Alternatively, you might prefer to keep the macro with all the other macros within this tutorial series; you can move it later if you change your mind.

```
Public Function GetFldrNames(ByRef Fldr As Folder) As String()

  ' * Fldr is a folder. It could be a store, the child of a store,
  '   the grandchild of a store or more deeply nested.
  ' * Return the name of that folder as a string array in the sequence:
  '    (0)=StoreName (1)=Level1FolderName (2)=Level2FolderName  ...

  ' 12Oct16  Coded
  ' 20Oct16  Renamed from GetFldrNameStr and amended to return a string array
  '          rather than a string

  Dim FldrCrnt As Folder
  Dim FldrNameCrnt As String
  Dim FldrNames() As String
  Dim FldrNamesRev() As String
  Dim FldrPrnt As Folder
  Dim InxFN As Long
  Dim InxFnR As Long
```

```
  Set FldrCrnt = Fldr
  FldrNameCrnt = FldrCrnt.Name
  ReDim FldrNamesRev(0 To 0)
  FldrNamesRev(0) = Fldr.Name
  ' Loop getting parents until FldrCrnt has no parent.
  ' Add names of Fldr and all its parents to FldrName as they are found
  Do While True
    Set FldrPrnt = Nothing
    On Error Resume Next
    Set FldrPrnt = Nothing   ' Ensure value is Nothing if following statement fails
    Set FldrPrnt = FldrCrnt.Parent
    On Error GoTo 0
    If FldrPrnt Is Nothing Then
      ' FldrCrnt has no parent
      Exit Do
    End If
    ReDim Preserve FldrNamesRev(0 To UBound(FldrNamesRev) + 1)
    FldrNamesRev(UBound(FldrNamesRev)) = FldrPrnt.Name
    Set FldrCrnt = FldrPrnt
  Loop

  ' Copy names to FldrNames in reverse sequence so they end up in the correct sequence
  ReDim FldrNames(0 To UBound(FldrNamesRev))
  InxFN = 0
  For InxFnR = UBound(FldrNamesRev) To 0 Step -1
    FldrNames(InxFN) = FldrNamesRev(InxFnR)
    InxFN = InxFN + 1
  Next

  GetFldrNames = FldrNames

End Function
```

3.2 Referencing a default folder

In `TestDefaultFldr()` I set `Fldr` to the default Inbox. The constant `olFolderInbox` can be replaced by other values giving access to any of the default folders. If you type `Set Fldr = Session.GetDefaultFolder(`, the VB editor will display a drop down list of all the possible values.

```
Sub TestDefaultFldr()

  Dim Fldr As Folder

  Set Fldr = Session.GetDefaultFolder(olFolderInbox)

  Debug.Print Join(GetFldrNames(Fldr), "|")

End Sub
```

On my laptop, `TestDefaultFldr()` displays `Outlook data file|Inbox` which came as a surprise. I wrote `GetFldrNames(Fldr)` to make sure that the folder I had referenced was the one I wanted. I had accessed the default Inbox and found it was empty! Store “Output data file” came with the default installation and I had ignored it since Outlook had created a store for each of my email accounts. It was only after discovering my empty default Inbox that I thought about how Outlook would know which of my email accounts was the account I would want as the default. Of the standard Outlook folders, there is either no default or the default is within “Output data file”. It may be possible to

change which Inbox is the default Inbox but I have not investigated because I am not sure which of my email accounts I would make the default if I did change. Just remember that all your Calendar Items, Tasks and so on are within “Outlook data file” and make sure you include “Outlook.pst” in your archive list.

Most Outlook objects have the property `Parent`. `GetFldrNames(Fldr)` records the name of the folder in an array before trying to access its parent. It loops adding names to the end of the array until it reaches the store. The store does not have a parent so the attempt to access it fails. The sequence of names in the array is reversed and then returned to the caller. I have used `Join` to turn the array of names into a displayable string.

3.3 Referencing any folder within any accessible store

`TestFldrChain()` demonstrates how to reference any folder within any accessible store:

```
Sub TestFldrChain()

  Dim Fldr As Folder

  Set Fldr = Session.Folders("A").Folders("A2"). _
                          Folders("A21").Folders("A213")

  Debug.Print Join(GetFldrNames(Fldr), "|")

End Sub
```

In `TestFldrChain()`: A is the name of a store; A2 is the name of a folder within A; A21 is the name of a folder within A2 and A213 is the name of a folder within A21.

What is happening here?

`Session` has a property `Folders` which is a list of all accessible stores.

`Session.Folders(integer)`, which I used in Part 2 of this tutorial, allows me to step through the stores in sequence when I do not know their names. `Session.Folders("A")` allows me to access a folder when I know its name.

`Session.Folders("A")` is a folder and it too has a property `Folders`.

`Session.Folders("A").Folders("A2")` gives me access to folder “A2” within store “A”.

I can chain as many `Folders("x")`s as necessary to reach any folder. If the chain is too long for one line, you can split the statement across several lines as I have.

Look for the most deeply nested folder within your installation and replace A, A2, A21 and A213 by the names of your store and folders. Increase or decrease the number of Folders in the chain as necessary.

If you update and run `TestFldrChain()`, it will output the following except that A, A2 and so on will have been replaced by your folder names:

```
A|A2|A21|A213
```

3.4 Listing the names of every folder within every accessible store

In Part 2, you were shown how to list every accessible store and the top level folders within each store. This involved a loop through the stores and then a loop for each store through its folders Above you have seen how to reference a known folder at any depth within the hierarchy of folders. This involved chaining together as many `Folders("x")`s as necessary to reach the folder.

I now want to list every folder, at any depth, within every store. The easiest coding technique for solving this type of problem where you must move down chains of varying lengths is **recursion**. If you are a serious programmer in another language or tool, you may already know about recursion. If you have ambitions to be a serious programmer, you will need to understand recursion eventually but not necessarily today. “Recursion” is one of those concepts that many find difficult to grasp at first. You can type “Recursion” into your favourite search engine and read the various attempts at explaining this concept. Alternatively, you can accept these macro work but not worry how they work.

Note the comment in `ListStoresAndAllFolders()`: these macros need a reference to “Microsoft Scripting Runtime”. Click `Tools` in the tab bar at the top of the VB Editor window then click `References`. You will get a list of all the available references (libraries). Some at the top will already be ticked. The remainder are in alphabetic order. Scroll down the list and click the box to the left of “Microsoft Scripting Runtime” to get a tick. Then click `OK`

```
Sub ListStoresAndAllFolders()

 ' Displays the name of every accessible store
 ' Under each store, displays an indented list of all its folders

 ' Technique for locating desktop from answer by Kyle:
 ' http://stackoverflow.com/a/17551579/973283

 ' Needs reference to “Microsoft Scripting Runtime” if "TextStream"
 ' and "FileSystemObject" are to be recognised

 Dim FileOut As TextStream
 Dim FldrCrnt As Folder
 Dim Fso As FileSystemObject
 Dim InxFldrCrnt As Long
 Dim InxStoreCrnt As Long
 Dim Path As String
 Dim StoreCrnt As Folder

 Path = CreateObject("WScript.Shell").SpecialFolders("Desktop")

 Set Fso = CreateObject("Scripting.FileSystemObject")
 Set FileOut = Fso.CreateTextFile(Path & "\ListStoresAndAllFolders.txt", True)

 With Application.Session
   For InxStoreCrnt = 1 To .Folders.Count
     Set StoreCrnt = .Folders(InxStoreCrnt)
     With StoreCrnt
       FileOut.WriteLine .Name
       For InxFldrCrnt = .Folders.Count To 1 Step -1
```

```
          Set FldrCrnt = .Folders(InxFldrCrnt)
          Call ListAllFolders(FldrCrnt, 1, FileOut)
        Next
      End With
    Next
  End With

  FileOut.Close

End Sub
Sub ListAllFolders(ByRef Fldr As Folder, ByVal Level As Long, ByRef FileOut As TextStream)

  ' This routine:
  '  1. Output name of Fldr
  '  2. Calls itself for each child of Fldr
  ' It is designed to be called by ListStoresAndAllFolders()

  Dim InxFldrCrnt As Long

  With Fldr
    FileOut.WriteLine Space(Level * 2) & .Name
    For InxFldrCrnt = .Folders.Count To 1 Step -1
      Call ListAllFolders(.Folders(InxFldrCrnt), Level + 1, FileOut)
    Next
  End With

End Sub
```

After you have run `ListStoresAndAllFolders`, there will be a new file on your DeskTop named “ListStoresAndAllFolders.txt” which will contain the promised list of stores and folders.

3.5 Moving a folder from one parent folder to another

Why do I want to reference a folder? In the next part I will show you how to access emails within a referenced folder. Here I will show you how to move a folder. I created a folder named “Test” within my Inbox. In `TestMoveFolder()`, I replaced “A” with the name of the store containing my Inbox. Running `TestMoveFolder()` moved “Test” to “Deleted Items”.

```
Sub TestMoveFolder()

  Dim FldrDest As Folder
  Dim FldrToMove As Folder

  Set FldrToMove = Session.Folders("A").Folders("Inbox").Folders("Test")
  Set FldrDest = Session.Folders("A").Folders("Deleted Items")

  FldrToMove.MoveTo FldrDest

End Sub
```

3.6 What you should remember from this part of the tutorial

- How to reference a default folder and the possible limitations of this technique.
- How to reference any single folder at any depth within any accessible store.
- How to display the full name of a referenced folder.

- How to reference one of the many, many available libraries that provide functionality beyond the default set of subroutines and functions.
- How to display the name of every folder within every accessible store.
- How to move a folder from one parent folder to another.

Read Introduction Part 3: Stores and all their folders online: https://riptutorial.com/outlook-vba/topic/8874/introduction-part-3--stores-and-all-their-folders

Credits

S. No	Chapters	Contributors
1	Getting started with outlook-vba	Community, Tony Dallimore
2	Introduction Part 1: Gaining access to Outlook's Visual Basic Editor	Tony Dallimore
3	Introduction Part 2: Stores and top-level folders	Tony Dallimore
4	Introduction Part 3: Stores and all their folders	Tony Dallimore