The VBA Help

VBE_Extras – user guide

For VBE_Extras version 1.7.2.0

Contents

What is VBE_Extras

VBE_Extras is a COM AddIn for the Visual Basic Editor (VBE) within 32- and 64-bit versions of Excel, Word, Outlook †, PowerPoint and Access (the 'host applications') - one installation works for all host applications. It works only on Windows 7 (or newer) and requires .NET Framework 4.8.

Once VBE_Extras is installed (see Installation), start one of the relevant host applications and open the VBE (Alt + F11, or using the 'Visual Basic' button in the 'Developer' tab of the ribbon). The VBE_Extras Commands are then available to you via a number of keyboard shortcuts, a Toolbar and via various menu items which are added:

- To the main VBE menu
- To the code window context menu
- To the Immediate window context menu
- To the Project window context menu
- To the UserForm Control context menu

For users of Excel, VBE_Extras is supported by the VBE_Extras Helper AddIn which can be downloaded and installed separately.

† see Appendix 5 – 'new Outlook'

Why use VBE_Extras

To save you time and help you write better code.

VBE_Extras provides a number of Commands that assist developers to create and edit VBA code and the number and scope of those commands are continually expanding over time. However, it is also the intent not to directly duplicate commands with those already available in certain other tools that also extend the capabilities of the VBE, notably the excellent MZ-Tools which also provides a wide range of VBE commands and enhancements (note that I have no link or association with MZ-Tools or its author).

And, unlike some other VBE AddIns, VBE_Extras will not slow down the process of creating code with regular lengthy "code parsing" exercises which block you from interacting with the VBE. Rather VBE_Extras will sit quietly in the background taking up zero resources (well, close to it … we have to log the cursor position and Modules being shown/edited for the Show History and Project Picker / Module Picker commands) until you ask it to do something.

Installation

VBE_Extras is installed on a per-user basis and does not require admin rights to install.

To download, click the 'download' link on The VBA Help website at https://www.thevbahelp.com/vbe-extras-download.

To install:
- Ensure you are the only user logged on to the device
- Close all MS Office applications (and any other applications on your device that can host VBA code)
- Unzip the downloaded file and then double-click on the VBE_Extras_Setup.exe file
- Follow the instructions in the installation wizard including accepting the licence

If you experience any problems while installing, please see Appendix 1 – Installation troubleshooting

Uninstallation

If you just want to temporarily suspend VBE_Extras, in the VBE main menu select Add-Ins > Add-In Manager > VBE_Extras then:
- Remove the tick (check) from 'Loaded/Unloaded' if you want to suspend VBE_Extras for this one session, it will load automatically in future
- Remove the tick (check) from 'Load in Startup' if you want to suspend VBE_Extras from automatically loading in future

If you want to permanently uninstall VBE_Extras then:
- Go to Control Panel > Programs and Features, or
- Go to Settings > Apps
- Then select VBE_Extras and then click the Uninstall button … you can choose to not delete your settings if you want them to be available in the future

Terminology

In this document:

This …	**… means this**
CCA	A Conditional Compilation Argument ... see Conditional Compilation Arguments
Declaration	A declaration of any type e.g. variables (declared with the Dim keyword), Procedures (declared with the Sub or Function keyword), Properties (declared with the Property Get/Let/Set keywords), Modules (no specific declaration keyword, visible in the Project window), Enums (declared with the Enum keyword) etc.
Field	A Module-level variable
Host / host application	The application that is currently hosting the VBE (Excel, Word, Outlook, PowerPoint or Access)
Project	A specific VBA Project

Project reference	Either:
	• A reference between two VBA Projects which can be:
	  ○ To a "child" Project (the active Project has a reference to the "child" Project)
	  ○ From a "parent" Project (the "parent" Project has a reference to the active Project)
	  ○ An "indirect" Project reference (the active Project and another Project both have "child" references to the same 3rd Project … the "indirect" reference is to other Project having a reference to the same "child" Project)
	• A reference from the active Project to a Type Library

Reference	Any reference to (or 'usage of') a declaration … a reference can be a: • "qualified reference" – the reference 'target' is preceded by the Module and/or Project name (separated by a '.') where relevant • "unqualified reference" – VBA allows you to refer to a member of another Module or Project without using the name of the Module and/or Project preceding it (delimited by a '.') … where a reference omits the Module and/or Project name then this is an 'unqualified reference'

type (lowercase 't')	The value type assigned to a variable, Function or other declaration (i.e. can be defined by the word(s) following the 'As' keyword or can use one of the Type hint characters or can be defined by one of the DefType statements)
Type ('uppercase 'T')	A 'user-defined Type' (UDT)
Type Library (or TypeLib)	A file that contains one or more 'types'. Types include Classes, Modules and Enumerations, Variables, Functions etc. Typically, you add a Project reference to a Type Library to enhance the functionality of VBA / the VBE … for example, you can add a Project reference to the 'Microsoft Scripting Runtime' Type Library to use the FileSystemObject and Dictionary Classes. To read more about Type Library's, see my Adding / updating "Project References" in the VBE blog post.
UI	The user interface i.e. of the host application

Language

As a solo developer who speaks only English, I'm afraid that I have developed VBE_Extras to display messages and accept inputs in the English language only. However, other than the few exceptions specified throughout this document, all functionality will work no matter which language you use as your Office 'display' or 'authoring' language.

What's new in VBE_Extras

See Appendix 3 – Update history

Limitations / exclusions

Finding references in code comments and string literals

It is the aim of VBE_Extras to accurately find every reference to a declaration in the code of a Project … no false positives (where the same name is used in the same scope) and no false negatives (missing references).

However, the same is just not possible in code comments and string literals. VBE_Extras just 'does its best'. There are a number of reasons why finding every reference in in code comments and string literals is not possible:

- Code comments and string literals can 'refer to' declarations from a scope in which that declaration would not normally be accessible, for example you can have code comments in one Procedure that refers to a variable 'Dimmed' in another Procedure … this is, of course, entirely 'legal' as far as VBA is concerned, but does mean that the link from that code comment to the variable Dimmed in another Procedure cannot be established
- Code comments and string literals can be used 'unqualified' (see Terminology) and without the usual 'context' that using the same name in code provides
- It is not possible to disambiguate between multiple declarations with the same name when used in code comments or string literals

Keywords used for Type element names

VBE_Extras cannot distinguish these from normal uses of the reserved word. Still, why would you want to call a Type element 'For' or 'Next' … you want to write code that is clear, right!

Late-bound references

To resolve references to a specific declaration, VBE_Extras requires declarations to be explicit about the value type that a variable holds / Function returns etc. If a variable is defined 'As Object' or 'As Variant' (late binding) as opposed to 'As SomeClass' (early binding) then VBE_Extras cannot determine the specific value type that you actually intended … in just the same way that the VBE itself cannot determine the specific value type.

Wherever possible, you should declare and return explicit value types (i.e. use early binding) so that VBE_Extras can help you work with your code.

Alternatively, consider using Special binding.

Using VBE_Extras in Access

Limitations / exclusions

When using VBE_Extras in Access, the following limitations / exclusions apply:

1. When parsing code (working with references, including identifying Orphan declarations), the following are excluded:
    a. Access Macros (including those referenced from Form, Report and Control events and those that reference to Functions or other VBA members)
    b. Expressions in Form, Report and Control events
    … so, for example, if you get References for code at cursor for a VBA Function that is referenced by an Access Macro, the list of references displayed will not include the reference in the Access Macro.
2. Go to declaration for code at cursor for a Control will only go to the Form/Report and will not highlight the specific Control
3. When working with an AddIn (when it is loaded as an AddIn e.g. via Database Tools > Add-ins), VBE_Extras will not identify Controls in a Form or Report and so cannot 'go to' them, rename them and will not correctly identify event handlers as being related to them. This functionality will work correctly when the AddIn is 'loaded directly' (e.g. via File > Open).
4. When working with a VBA Project that is open because it is referenced by a 'parent' Project (and so it is not the Project associated with the Database that is loaded in the Access UI), then:
    a. The same restrictions apply as for an AddIn (see point 3)
    b. That Project cannot be saved (and so updates made by VBE_Extra's refactoring commands cannot be saved) … see Referenced VBA Projects (including the 'Access workaround') for more info
5. See the 'Scope' section of Version Control for what is included in Version Control exports / imports (for example, doesn't include QueryDefs, Macros, Tables)

Other

When renaming a Project or Module, Access allows certain characters to be included in the name that are 'not recommended' by Microsoft (see https://learn.microsoft.com/en-us/office/troubleshoot/access/error-using-special-characters). VBE_Extras will warn you when one of these characters is used in a proposed new name.

Using VBE_Extras with files stored on OneDrive

VBE_Extras works well with files (Workbook, Documents, Databases etc) stored on OneDrive … unless you have a non-standard OneDrive setup.

When carrying out many commands, VBE_Extras needs to know the file path of the file it is working on (for example, for Version Control when using 'Project folder' or 'Project sub-folder' as the export / import folder). To get this location, VBE_Extras uses the methods and properties supplied by the host application.

However, as has been detailed in various https://stackoverflow.com/ questions and answers, when a file is saved on OneDrive, the host application may return a file path ("C:\ …") or may return a web link ("https://..."). When a web link is returned, VBE_Extras can parse this into a file path so long as you have a standard OneDrive setup. An example of a non-standard OneDrive setup include having multiple OneDrive accounts on the same device where the 'personal' folder for one account is inside the 'personal' folder for the other account.

If you have a non-standard setup and the file path could not be parsed then you may see messages advising you of this. The solution, if you want to continue using VBE_Extras, is either to fix your OneDrive setup (which might not be simple … once it has installed on your device) or to move the file to a local (i.e. non-OneDrive) folder.

Code in the Immediate window

The VBE_Extras commands work with code in the code pane (i.e. the main window in which you edit code) and do not work with code in the Immediate (or any other) windows.

Inclusions

Referenced VBA Projects

VBE_Extras is 'multi-Project'. This means that all commands (except where specified; in host applications that allow it), operate on the active Project plus other VBA Projects that are connected to the active Project by a 'Project reference' ('Project reference' means a reference from a 'parent' VBA Project to a 'child' VBA Project via the VBE's References dialog: Tools > References).

So, for example, VBE_Extras can:
- Find declarations in other Projects
- Find references in other Projects
- Rename declarations (and all references to that declaration) in other Projects

Some commands are intended to be 'single-Project', these are identified in the menus by use of the 'this Project' text, for example:
- 'Tasks in this Project'
- 'Previous Lists for this Project'

Note that being multi-Project does not mean that a command always operates on every open VBA Project – VBE_Extras can determine:
- Which Projects the active Project has a reference to (ie 'child' Projects)
- Which Projects have a reference to the active Project (ie 'parent' Projects)
- Which Projects also have a reference to a child Project that the active Project also has a reference to (ie 'indirect' Projects)

… and carries out commands on those Projects only and, even then, does so only if appropriate (eg if you ask VBE_Extras to rename a Public variable in a standard Module that does not have 'Option Private Module' then it will look for references in the active Project and all 'parent' Projects but if you ask VBE_Extras to rename a Private variable then it will look for references only in the active Project).

Obviously, VBE_Extras can only operate on open Projects. The host application will always automatically open any 'child' Projects (so they will always be updated) but will not automatically open any 'parent' or 'indirect' Projects (so they will not always be updated … you need to open them before using any 'refactoring' commands). If the active Project has 'broken' references to other Projects then those other Projects cannot be operated on.

Additionally, VBE_Extras can only operate on unlocked Projects – if a Project is locked (Tools > Project Properties > Protection > Lock Project for viewing) then the VBE itself prevents VBE_Extras from reading or updating code in those Projects.

Note that not all supported host applications work in the same way when it comes to references to other Projects:
- Excel and Word allow references to other Projects and code in those Projects can be read, updated and saved
- Access allows references to other Projects and allows code in those Projects to be updated – but the VBE itself (when hosted in Access) does not allow those other Projects (i.e. Projects other than the one that is associated with the Database that is open in the Access UI) to be saved … note that both the 'Save' icon and 'Save' menu item in the VBE's menu will be disabled when the Project that is active in the VBE is an 'other' Project … see the 'Access workaround', below
- PowerPoint allows references only to AddIn Projects and the code in AddIns is not available to be read or updated (this is not a restriction of VBE_Extras, rather it is a restriction imposed by PowerPoint itself)
- Outlook does not allow references to other Projects

In the Settings you can choose to switch off 'multi-Project' so that VBE_Extras ignores all referenced Projects. You can also choose to block individual Projects based on their Project name, file name or full path (none of these are case-sensitive). Code in blocked Projects will not be read or updated in the same way that code in locked Projects will not be read or updated.

Access workaround

When the VBE is hosted by the Access application, the VBE cannot save a 'child' Project, only the active / 'parent' Project can be saved. This is a limitation of the VBE itself, not of VBE_Extras.

As such, if you need to update code in both the active (i.e. 'parent') Project and in a 'child' Project in Access (e.g. if you need to rename a declaration that is declared in the child Project but is referenced from the active/parent Project), the workaround is:

1. It should go without saying … but, first, make backup copies of all Projects to be updated
2. With the parent Project as the active Project, use VBE_Extras to rename the declaration/reference
3. Save the parent Project and close it in the Access UI (meaning that the changes to the parent Project are saved but the changes to the child Project are not saved)
4. If you have multiple parent Projects each using the declaration that is declared in the child Project then update each parent Project (points 1 and 2) before continuing to update the child Project
5. Open the child Project directly in the Access UI then open the VBE and carry out the same rename (i.e. use VBE_Extras to rename the same declaration/reference to the same name)
6. Save the child Project and close it in the Access UI (meaning that the changes to the child Project are now also saved and align with those in the parent Project)
7. Re-open the parent Project and check it compiles correctly (i.e. Debug > Compile)

'Square bracket' references

In VBA, any reference can be enclosed in square brackets whether the square brackets are required or not (they are only required where the name of the reference includes normally-invalid characters … otherwise, square brackets are allowed but are redundant).

Outside of Access, square brackets are rarely used and typically only for Enum member names that are hidden or include normally-invalid characters. However, VBE_Extras can handle square bracket references in Access (for example when Module names or Control names include normally-invalid characters) and all other host applications including when you Rename code at cursor for Enum members, Modules and Controls.

The bang ("!") operator

If you're not familiar with the bang operator (likely you don't code in Access), the bang operator provides late-bound access to the default member of an object, by passing the literal name following the bang operator as a String argument to that default member.

VBE_Extras can handle the bang operator. However, as 'code' following (i.e. to the right of) the bang operator is late-bound (being passed as a String to the default member of the object on the left of the bang operator) then references are identified as Strings (not code) … because they are (Strings, that is). In the same way that the VBE itself cannot compile-time check the String following the bang, VBE_Extras cannot parse it as code.

If you want your code to be clear and increase its maintainability, avoid the bang!

Type hint characters

VBE_Extras can handle type hint characters e.g. a variable defined such as "Dim text$" instead of "Dim text As String" … VBE_Extras handles all 7 type hint characters: % for Integer; & for Long; @ for Currency; ! for Single; # for Double; $ for String; ^ for LongLong (on 64 bit hosts).

While VBE_Extras handles type hint characters, I don't want to encourage their use … they're from the 1980s and should stay there.

DefType statements

DefType statements are parsed, determining the value type of variables, Functions, Property Gets etc for the relevant letter / letter groups. Note that malformed DefType statements (i.e. that will not compile) are ignored.

Please don't use DefType statements … every declaration deserves an 'As'.

Declarations with the same name in the same scope

It is possible to do some truly weird stuff in VBA. VBE_Extras can handle the (perfectly legal, though shocking unreadable) code to the right. Yes, you can (though you shouldn't!) declare:

- A Type, an Event and any one other type of Module-level declaration with the same name all in the same Module
- A Module and a Module-level declaration in the same Module with the same name
- A Sub or a Property Let/Set can have a parameter, variable, static or constant within it with the same name
- A compiler Constant can have the same name as another Module-level declaration
- An Enum and an Enum member can have the same name
- A Type and a Type element can have the same name
- A line label can have the same name as another declaration within the same Procedure/Property, with the same name as the Procedure/Property itself, and with the same name as the Module the Procedure/Property is within

While VBA allows it, nobody wants code like "ThisThing"?! But if you do have code like this, VBE_Extras can parse the code and get the right references to the right declarations.

```
' Class Module name: "ThisThing"

Option Explicit

Event ThisThing(ThisThing As Long)

Private Type ThisThing
    ThisThing As Long
End Type
```

```
Sub ThisThing()
    On Error GoTo ThisThing

    Dim ThisThing As Long
    ThisThing = 1

    Dim t1 As ThisThing
    t1.ThisThing = 2

    Dim t2 As ThisThing.ThisThing
    t2.ThisThing = 3

    RaiseEvent ThisThing(ThisThing)
    RaiseEvent ThisThing(t1.ThisThing)
    RaiseEvent ThisThing(t2.ThisThing)

    Debug.Print "We are here"
    Debug.Print ThisThing / 0
    Exit Sub

ThisThing:
    Debug.Print "And now we're here"
End Sub
```

Shapes

In Excel only, a Sub or Function in a standard or document Module can be assigned to a Shape such that it runs when the Shape is clicked … the assignment of a Sub or Function is stored in the Shape's OnAction property.

VBE_Extras treats the OnAction property of a Shape in Excel as a reference to a declaration (i.e. the declaration of the Sub or Function). The Shape will be included in the references for that declaration.

The Project must have been saved at least once for Shapes to be correctly recognised.

Requirements

Option Explicit

Much of the functionality of VBE_Extras (for example finding declarations, references, identifying orphans, renaming) requires each variable to be explicitly declared. If VBE_Extras cannot find a declaration, it does not 'assume Variant'.

Good programming practice requires all variables to be explicitly declared. If you use VBE_Extras in a Project that has one or more Modules without Option Explicit, you will see warning dialogs (you can change the frequency of the warning in the Settings).

You can also consider using the Make 'Option Explicit' command.

Saved Projects

Much of the functionality of VBE_Extras requires the active Project to have been saved at least once.

Some functionality requires for there to be no more than one open 'never-saved' Project (where 'never-saved' means, for example in Excel, where a new workbook has been added and it has never been saved … this does not mean a workbook that was re-opened from a file).

And some functionality requires for there to be no open 'never-saved' Projects.

There are various reasons for this, however, it boils down to the fact that VBE_Extras needs to interact with the VBE and the VBE is not helpful when it comes to open 'never-saved' Projects:

- Saving a Project generates a (unique) file name for the Project and a unique identifier is required for many of the commands of VBE_Extras … when there is only 1 open never-saved Project then VBE_Extras can use the 'lack of unique identifier' as the actual identifier … but this doesn't work when there are 2 or more open never-saved Projects
- Various properties of the VBE stop responding correctly (they return errors even when read-from, not written-to) when there is one or more open never-saved Project(s) and this state of returning errors persists even when the open never-saved Project(s) are closed or saved (the 'error state' persists until the host application is closed and re-started) … hence the workaround is to not to access those properties when there is one or more open never-saved Project(s) – this is the reason why, for example, VBE_Extras will not show a UserForm designer when there is one or more open never-saved Project(s)

Conditional compilation

VBE_Extras is 'conditional compilation aware'. Based on any compilation constants (both declared globally in the 'Project Properties' dialog and declared in a Module) and compilation directives (#If … #Then … #Else), VBE_Extras will work out which lines and, therefore, which variables, Constants, Procedures etc are 'in scope' for compilation or not.

VBE_Extras correctly determines expressions in #If and #ElseIf directives for:

- Win16 †, Win32, Win64, VBA6, VBA7, Mac – determined appropriately for the device
- Boolean, numeric, date/time and string operations
- Computes operators (+, -, *, /, \, Mod, ^ and &), comparators (=, <>, <, >, <=, >= and Like) logicals (Not, And, Or, Xor, Eqv and Imp … logical comparisons only, not bitwise comparisons), handles parenthesis, interprets hex &H and oct &O and works with VBA Functions (for example, Int(), CLng(), Abs())

VBE_Extras does not correctly determine expressions in #If and #ElseIf directives that use:

- Global Conditional Compilation Arguments unless you have instructed VBE_Extras to read these from the Project Properties window using the Conditional Compilation Arguments commands. Note these are read only once when the menu item is selected, not every time any VBE_Extras command is called (on the basis that these are not regularly updated, then there is not a need to do so).

VBE_Extras supports two modes of behaviour with respect to conditional compilation:

- Match to conditional compilation directives … in this mode, for example, declarations and references that are not currently 'in scope' will not be shown in dialogs
- Do not match to conditional compilation directives … in this mode, for example, declarations and references that are not currently 'in scope' will be shown 'greyed-out' in dialogs

The mode being used can be set in the 'Other' tab of the VBE_Extras Settings.

There are two exceptions that ignore the mode:

- The Rename code at cursor command will display a dialog asking whether you want to rename the 'out of scope' references as well as the 'in scope' references
- The Orphan declarations commands always ignore declarations and references on excluded lines

† Always evaluated to False assuming all VBA code is now developed and run on 32 or 64 bit devices

Commands

Commands available in the main menu

Many of these commands are also available in other menus, in the Toolbar and by using shortcut keys.

The VBE_Extras main menu uses the 'x' key as its accelerator. If you need to change this, see Appendix 4 – changing the main menu's 'x' accelerator key.

Info for code at cursor

The result of this command depends on the position of the cursor at the time. When the cursor is on:

- A declaration, the result will be information for that declaration, the same as Show declaration for code at cursor
- A reference, the result will be information for the declaration that reference relates to, again the same as Show declaration for code at cursor
- A Module (document, standard, Class or UserForm) name, the result will be a list of all declarations in that Module
- White space, the result will be a list of all declarations in the Module that is active in the code window

'Info for code at cursor' is also 'Type Library aware' (and works with Special binding) – for example if you declare the variable `fso` (or whatever name you choose) `As Scripting.FileSystemObject` and then type `fso.CopyFile` and place the cursor on `CopyFile` and trigger 'Info for code at cursor' you will see this dialog.

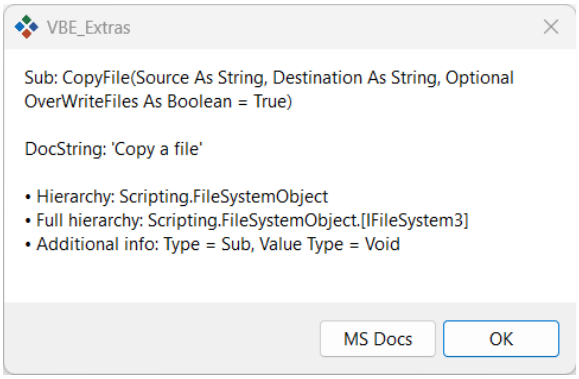


Figure 1 - Special binding and 'Info for code at cursor'

For some Type Libraries, both when using early binding and when using special binding, VBE_Extras can open the relevant MS Docs web page for further information. See MS Docs for details.

References for code at cursor

If the cursor is on a declaration or reference to a declaration, then this will display a dialog with a list of all references of that declaration (including the declaration itself). The dialog will allow you to show or hide references in code comments, in strings and 'return values' (when a value is being set to the name of a Function or Property Get to be returned to the calling code). The dialog will show a column displaying the Project that each reference is within only if the references are contained within multiple Projects.

VBE_Extras does not simply find all occurrences of a name, rather if there are multiple Declarations with the same name in the same scope then it can apply code-parsing rules to determine which references relate to which declaration.

This command will also find implementations of Interfaces, handlers of Events including for Controls on UserForms (and Reports and Forms in Access).

Note that VBE_Extras tests whether Class, Document and UserForm Modules have their VB_PredeclaredId attribute set to True, so allowing for 'static' access of members, and tests whether Class, Document and UserForm Modules have their VB_Exposed attribute set to True, so allowing other VBA Projects that reference the Project to 'see' the Module and its members. See also Set VB_PredeclaredId attribute and Set 'VB_Exposed' attribute.

You can control whether the declaration (or declarations if conditional compilation is involved) is shown at the top of the list or shown in the default order (being Project, Module-type, Module, line, column) in the Settings.

The list of references also includes information on how the reference is being used (i.e. Get, Let or Set) in the 'Use' column. Note that some references are neither a Get, Let nor a Set e.g. the reference of a declaration (other than constants and parameters), references of Modules, references in Strings and code comments etc.

Excel only: Shapes that have an OnAction property that references a Sub or Function in a standard Module will be included in the list of references.

Access only: see Using VBE_Extras in Access

Highlight

Highlights on-screen matches of declarations/references, words, parenthesis, speech-marks, or 'text' … allowing you to quickly identify those matches.

- If the cursor is positioned within a declaration or reference (whether it is an 'insertion point' or expanded to one or more characters within the name of the declaration or reference), all of the matching on-screen declaration(s) and reference(s) will be highlighted
- Otherwise, if the cursor is an insertion point within a 'word' (whether a VBA keyword or not) that is not in a string literal or in a code comment, all of the on-screen matches of that word will be highlighted
- Else if the cursor is expanded to one character that is an opening or closing parenthesis or an opening or closing speech-mark, the matching partner of the parenthesis or speech-mark will be highlighted
- Otherwise, if the cursor is expanded to one or more characters, all of the on-screen matches of that range of character(s) will be highlighted

There are two types of highlighting:

- New-style … temporarily draws a highlighting rectangle over each match - normally a 'solid' rectangle, however, if the reference is 'compiled-out' due to conditional compilation then the rectangle will be 'empty'
    - New-style highlighting only works for mono-spaced fonts (if you want to confirm which fonts work with new-style highlighting, use the Themes dialog), and
    - When the code pane window is not 'split' (but does work when multiple code pane windows are being shown e.g. using 'tiling')
- Old-style … temporarily renames the references to a series of fixed characters

In the Settings, you can control:

- Which style is used (new, old or 'try new, fallback to old') and, if old, the colour used
- Whether references are highlighted even if they are 'compiled-out' and the Conditional compilation mode is set to 'Match to conditional compilation directives' (i.e. when such 'compiled-out' references would normally be ignored)
- The duration of the highlight
- The line 'breadth' used to form the 'empty' rectangle for new-style highlighting of declarations and references if the reference is 'compiled-out' due to conditional compilation

Show All Members

The aim of 'Show All Members' is to improve on the VBE's Intellisense 'auto list members' dialog. This dialog is shown, typically, when you type a '.' following the name of a declaration that has child-members … for example, following the name of a Module in your VBA Project then you will see a list of the (non-Private) members of that Module; or following the name of an object such as an Excel Worksheet or a Word Document then typing a '.' will show you the (non-Private) members of that object. Note that …

- Whether the 'hidden' members of that object are shown is controlled by the VBE's built-in show/hide hidden members setting … however, you can also control this directly from VBE_Extras … see Show / hide hidden members
- Whether the 'restricted' members of that object are shown is controlled by the 'Show restricted members' setting in the 'TypeLibs' tab of the Settings

The big downside of the VBE's dialog is that it's not very user-friendly:

- It's very small (showing around half a dozen members at a time)
- It's not searchable
- It shows the name of the member (and the type as an icon) only
- When you display the dialog without a parent (i.e. without a '.' … by pressing Ctrl + J or using the 'Edit' > 'List Properties / Methods' menu item) then every available member is displayed in one big 'flat hierarchy' list (including members from within your VBA Project as well as members from each Type Library that your Project has a Project reference to)

The 'Show All Members' command is designed to fix this by displaying an improved dialog that:

- Is much larger (and can be re-sized)
- Is searchable (including "?" and "*" wildcards … see Appendix 8 – wildcards in 'filter text boxes') and filterable by member-type
- Shows not only the name of the member but also its type and its full 'signature' - and, when members being displayed come from both a Type Library and from your VBA Project, a column indicating in which of those two the member is declared (this latter can occur, for example, in ThisWorkbook in Excel, in Forms/Reports in Access, and in UserForms)
- Allows access to the MS Docs web pages for each members (for certain Type Libraries only … see MS Docs for more details) via the right-click / context menu
- When you display the dialog without a parent (i.e. by pressing Ctrl + . or using the 'Extras' > 'Show All Members' menu item while the cursor is in 'whitespace') then every available top-level member is displayed (including members from within your VBA Project as well as members from each Type Library that your Project has a Project reference to) … but if you prefer the VBE-style 'flat hierarchy' then you can enable this in the 'TypeLibs' tab of the Settings
- Allows you to just copy (rather than insert) the name of the member if you prefer
- If the member is a Constant or an Enum member from within a Type Library, then you can optionally insert or copy the value (rather than the name) of the Constant or Enum member … this is useful if using late binding or Special binding (when 'copy value' is available, the 'Copy' and/or 'Insert' buttons of the Show All Members dialog will automatically become 'split buttons' allowing the value to be copied or inserted)
- If the member has a 'doc string' then it will be shown to you if you hover the mouse pointer over the member (and if the 'Show doc string' option is selected in the Options for the Show All Members dialog) … this includes 'doc strings' defined:
    - In your VBA code … if you include lines with code comments immediately above, say, a procedure then they will be shown
    - In Type Libraries … though note many Type Libraries do not define 'doc strings' for their members … to see this in action for a Type Library, show the Show All Members dialog for an object in the Microsoft Scripting Runtime Type Library (e.g. the FileSystemObject or the Dictionary)

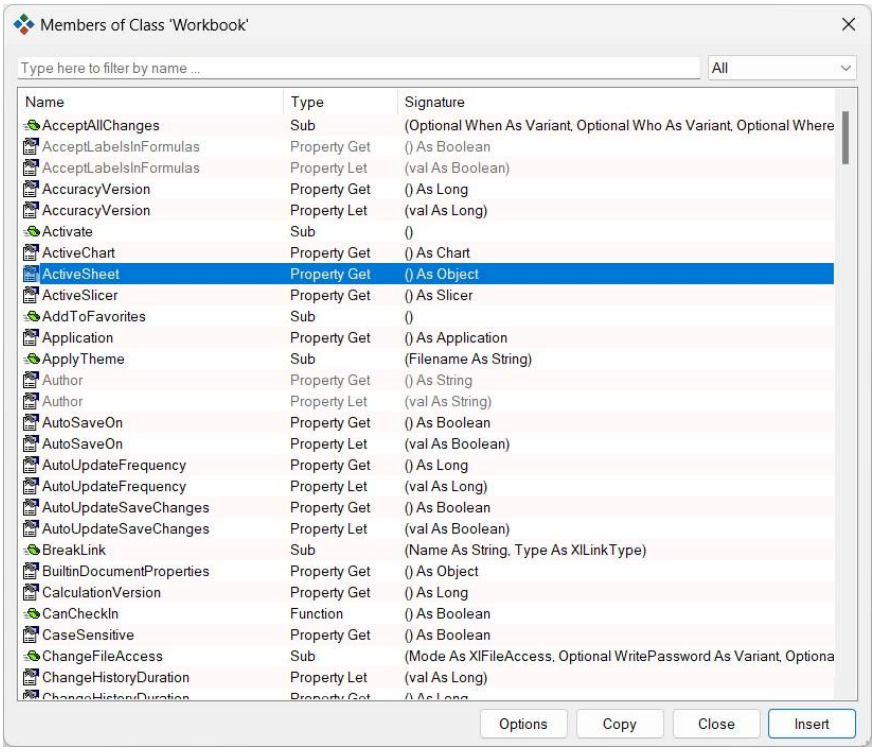


Figure 2 - The 'Show All Members' dialog

The Show All Members dialog can be shown in the following 'contexts':

- In whitespace … as noted above, all top-level members are shown (unless you have checked "Show members in a 'flat hierarchy'" in the 'TypeLibs' tab of the Settings
- Following a member … all child members of that member are shown (see Figure 2 above showing all child members of Excel's Workbook object)
- Following a comparator (=, <, >, <=, >=, <>) which follows the name of a variable (or Function, Property Get etc) which has the value type of an Enum (either an Enum defined in your VBA code or an Enum defined in a Type Library) … see Figure 3 where Ctrl + . has been pressed with the cursor following the '='

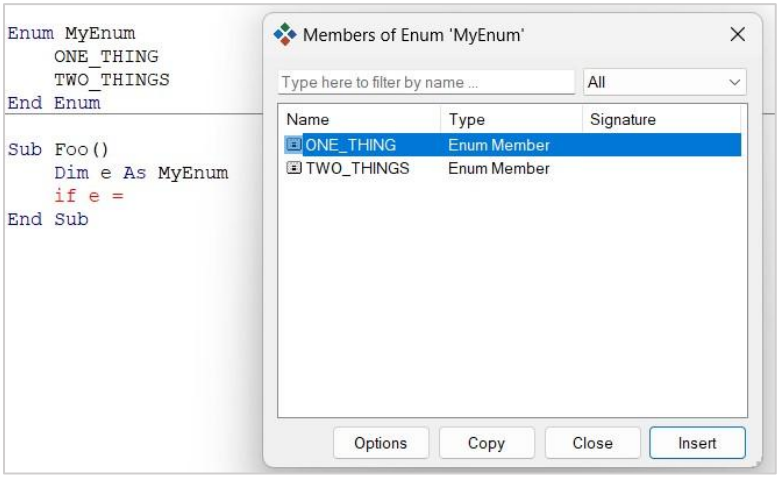


Figure 3 - Enum members

- Following an assignment operator (=) where the variable (or Property Get) being assigned to is declared with the value type of an Enum (again, either an Enum defined in your VBA code or an Enum defined in a Type Library) … similar to the above for comparators
- When passing an argument to a procedure or property and the respective parameter requires an Enum member … see Figure 4 where Ctrl + . has been pressed with the cursor following 'filename,'

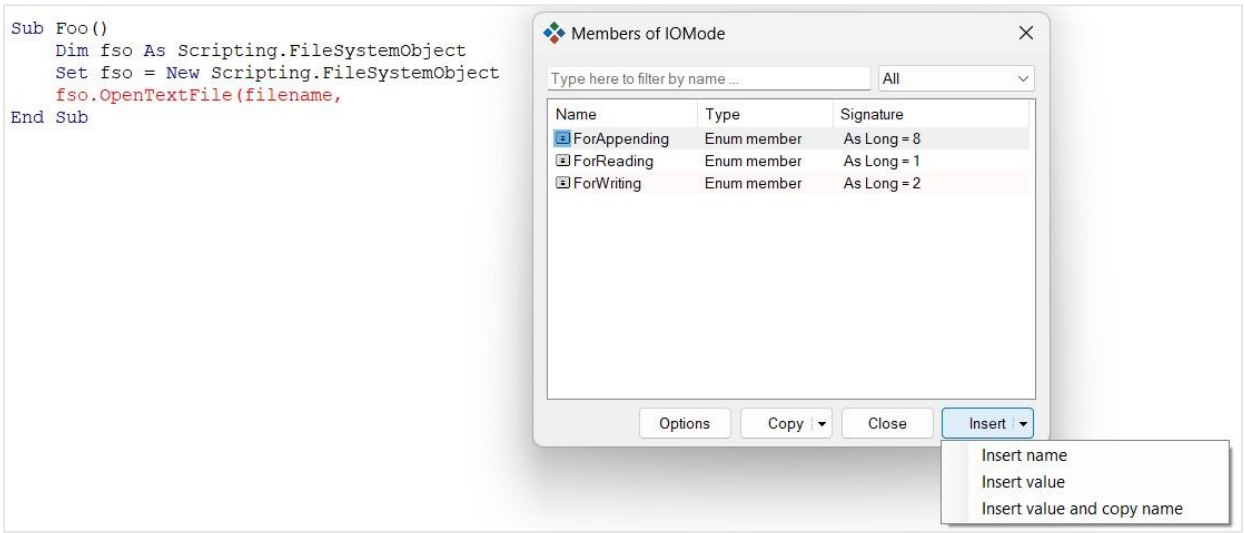


Figure 4 - Enum members as argument

Show All Members – things to know

If a member (or parameter etc) is declared as a Variant then Show All Members cannot show any members in the same way as the VBE cannot. Showing members depends on the explicit type being used. This applies when showing members for declarations in your VBA code and when showing members for declarations in Type Libraries … and I'm afraid that some Type Libraries declare far too many things as Variant … stand up, Excel!

Show All Members doesn't work for the 'Debug' object (i.e. as in `Debug.Print` or `Debug.Assert`) as Debug is not actually part of a Type Library, being built into the VBE itself.

As noted above, Show All Members uses a true hierarchy (not a 'flat hierarchy'). This means that for members of Modules and Enumerations, Show All Members considers those members not to be 'top level' … whereas the VBE does consider them to be 'top level'. For example, VBE's Err keyword is a Function within the Information Module of the VBA Library. If you type `Err` in the VBE, place the cursor after it and then press Ctrl + . you will not see a list of members, rather you will see a dialog telling you that Err isn't accessible. This is because Err is not 'top level'. If you use VBE_Extras 'Qualify reference' command on `Err`, it will offer to prefix it with `VBA.Information`. You can then press Ctrl + . and see its members.

If there are multiple members with the same name (e.g. a Property Get and a Let and/or Set) then they will each be shown in the Show All Members dialog whereas the VBE would just show one. You can choose to 'Collapse Properties' to show only one Property where there are multiple with the same name using the 'Options' button in the dialog (when you do this and there are multiple Properties with the same name then the 'Signature' is that of the Property Get).

If you Show All Members in whitespace with the "Show members in a 'flat hierarchy'" setting unchecked (the default), then the list shown to you will include: Type Library program names; VBA Project names (the active Project and any referenced Projects); Modules names in the active Project (Classes only if they have their VB_PredeclaredId attribute set to True … see Set 'VB_PredeclaredId' attribute); Enum names in the active Project; Module-level declarations in the Module that is being shown; procedure-level declarations in the procedure or property that the cursor is in.

Special binding

If you are aware of the differences between early and late binding and often develop your VBA code using the former but switch to the latter when deploying, then 'special binding' may be useful to you.

Using VBE_Extras' capacity to parse Type Libraries allows the Show All Members command (and other commands … noted below) to work without having a Project reference to the Type Library – see Figure 5.

```
Option Explicit

' <Ref>C:\Windows\System32\scrrun.dll</Ref> "Scripting"

Sub Foo()
    Dim fso As Object ' As Scripting.FileSystemObject
    Set fso = CreateObject("Scripting.FileSystemObject")
    fso
End Sub
```

Members of Class 'FileSystemObject'

Type here to filter by name ... | All

Name	Type	Signature
BuildPath	Function	(Path As String, Name As String) As String
CopyFile	Sub	(Source As String, Destination As String, Optional OverWriteFiles As Boolean = True)
CopyFolder	Sub	(Source As String, Destination As String, Optional OverWriteFiles As Boolean = True)
CreateFolder	Function	(Path As String) As IFolder
CreateTextFile	Function	(FileName As String, Optional Overwrite As Boolean = True, Optional Unicode As Bool...
DeleteFile	Sub	(FileSpec As String, Optional Force As Boolean = False)
DeleteFolder	Sub	(FolderSpec As String, Optional Force As Boolean = False)
DriveExists	Function	(DriveSpec As String) As Boolean
Drives	Property Get	() As IDriveCollection
FileExists	Function	(FileSpec As String) As Boolean
FolderExists	Function	(FolderSpec As String) As Boolean
GetAbsolutePathName	Function	(Path As String) As String
GetBaseName	Function	(Path As String) As String
GetDrive	Function	(DriveSpec As String) As IDrive
GetDriveName	Function	(Path As String) As String

Options | Copy | Close | Insert

Figure 5 - Special binding

Here you can see that `fso` is declared `As Object` (i.e. it is late bound) but Show All Members can still show all of its child members (in this case, Ctrl + . has been pressed following the `fso`). Three things need to be in place for this to work:

1. You need to have enabled 'special binding' in the 'TypeLibs' tab of the Settings.

2. Following the declaration (in this case `Dim fso As Object`), you need to tell VBE_Extras what the early-bound equivalent type would be, this is done as a code comment on the same line as the declaration – the code comment must include the word `As` and then the type … so here it is `As Scripting.FileSystemObject` (though `As FileSystemObject` would do - but including `Scripting` is better … as it is when using early binding … to correctly qualify the type and make the process of identifying which Type Library the type is in simpler and less prone to name clashes). This needs to happen after each declaration that you want special binding to work for.
3. At the top of the Module in which you want special binding to work, in the declaration lines (or, at least, before the first procedure or property is declared), you need to tell VBE_Extras which Type Library(s) you want special binding to be active for by specifying the full path in a code comment – in this case it is the "Scripting" library and the code comment is `' <Ref>C:\Windows\System32\scrrun.dll</Ref> "Scripting"` … the path must be enclosed in `<Ref>` tags (anything after the closing `</Ref>` tag is ignored but I find it useful to include the 'program name' for the Type Library as a reminder). You can get the relevant code comment for a specific Type Library using the Update Project References dialog, right-clicking on the relevant Type Library and selecting the 'Copy as <Ref>' menu item. This needs to happen once in each Module for each Type Library that you want special binding for.

In addition to Show All Members, other VBE_Extras commands that support special binding are:

- Info for code at cursor
- Qualify reference
- AutoText when adding a Select…Case statement using a variable, or other declaration with a value type, where that value type is an Enum from a Type Library
- Add Enum from Type Library
- Add a "GetEnumAsString()" Function

The early bound equivalent type code comment

You must include `As` in the code comment following the declaration of the variable that you want special binding to work for.

For variables (including Module-level variables), statics, Functions, Property Gets and Type elements, the `As Type` code comment should be at the end of the line on which they are declared. I recommend only one declaration on any one line unless you want to use the same `As Type` for multiple variables e.g.

```
Dim a As Object, b As Object, c As Object ' As Outlook.Folder
```

… will set the early bound equivalent type to Outlook.Folder for each of a, b and c.

For parameters, the `As Type` code comment should be in a line either above or below the procedure / property declaration and it should be preceded by the name of the parameter e.g.

```
' ---------------------------------------------------------------------------------------------------------------------
' Purpose: Do something
' Param fso As Scripting.FileSystemObject: A file system object that we will do something with
' Param dict As Scripting.Dictionary: A dictionary that we will do something with
' ---------------------------------------------------------------------------------------------------------------------
Sub DoSomething(fso As Object, dict As Object)
    ' code
End Sub
```

… will set the early bound equivalent type to Scripting.FileSystemObject for fso and to Scripting.Dictionary for dict.

The `As Type` can be enclosed in parenthesis if desired, for example the following is also valid:

```
' Param fso (As Scripting.FileSystemObject): A file system object that we will do something with
```

The `As Type` code-comment can be on a subsequent physical line (i.e. following a line-continuation character) if required, however, everything from the apostrophe starting the code-comment to the type must be on a single physical line.

Special binding – things to know

As noted above, you must include `As` in the code comment following the declaration of the variable that you want special binding to work for!

If a member (or parameter etc) is declared as a Variant or Object then special binding cannot show any members for it (or show 'Info for code at cursor' for it, or 'Qualify reference' for it) – special binding depends on explicit types being used.

All Project references have a specific priority order (note the order of Project references in the VBE's References dialog or in VBE_Extras equivalent Update Project References dialog) such that if multiple Type Libraries (or VBA Projects … whatever the reference is to) have a member with the same name, the reference with the higher priority 'wins'. The Type Library(s) "referenced" by <Ref> code comments are considered to have a lower priority than all actual Project references and are in the order they are specified in the Module (lowest line number = highest priority). If a single Type Library is referenced via an actual Project reference and via a <Ref> code comment then the <Ref> code comment will be ignored.

The path in the <Ref> code comment must be an absolute path (not a relative path).

Special binding works with hidden members of Type Libraries if you are showing hidden members (see Show / hide hidden members).

When using special binding, you can get direct access to the MS Docs web pages for objects and members of Type Libraries via the Info for code at cursor and Show All Members commands.

Command search

Search for any VBE_Extras command by name … view which menu it is in and its location within that menu. Also view any keyboard shortcut associated with it and whether it is present in the VBE_Extras Toolbar.

Note that if a command is in multiple menus (and many of them are), the menu shown is the 'most primary' menu of all of those that the command is in … that is, in order:

- The main menu
- The Project window menu
- The UserForm menu
- The UserForm Control menu
- The Code window menu
- The Immediate window menu

Note that:

- Some commands cannot be executed in every host application and/or for every 'Office display language' (such commands are greyed-out)
- Project Picker / Module Picker and AutoText cannot be executed from the "Command search" dialog as they require a certain key combination to be pressed

Declarations

Show declaration for code at cursor

If the cursor if on a declaration or reference, then you will be shown information for that declaration

List declarations in this Procedure / Property

Displays a dialog with a list of declarations in the Procedure or Property that the cursor is in.

List declarations in this Module

Displays a dialog with a list of declarations in the active Module (optionally, can select the types of declarations to show in the list).

List declarations in this Project

Displays a dialog with a list of declarations in the active Project (optionally, can select the types of declarations to show in the list).

List declarations referenced from other Modules in this Project

Displays a dialog with a list of declarations in the active Module that are referenced from any other Module of the same Project.

Excel only: for clarity, this excludes references from the OnAction property of Shapes (i.e. OnAction is not considered to be a reference from another Module).

Refactor

Rename code at cursor

If the cursor if on a declaration or reference, then you can rename that declaration and all references to it (including references in other open VBA Projects where those Projects have a reference to the Project containing the declaration … see Referenced VBA Projects for more info).

When renaming, VBE_Extras does not simply find and change all occurrences of a name, rather if there are multiple Declarations with the same name in the same scope then it can apply code-parsing rules to determine which references relate to which declaration and only the appropriate instances are changed.

If the declaration is a Property then any matching Property (i.e. Get, Let and Set) will be renamed. Also, VBE_Extras will offer to rename any backing Module-level variable or Type element where a match could be identified (to match, the variable or Type element must be referenced within the body of the Property and must have a matching name other than any prefix e.g. 'm' for Module-level, and both must be in the same Module).

If the declaration is a Getter (Function) or Setter (Sub) then any matching Getter or Setter will be renamed and again VBE_Extras will also offer to rename any backing variable or Type element with the same rules as above for a Property.

If the declaration is a backing member itself, then VBE_Extras will offer to also rename the related Properties or Getter/Setters – again applying the same rules. In addition, the following must be true:
- The backing member is a Module-level variable, or
- The backing member is a Type element and there is only one Module-level variable defined as that Type element's parent Type in the same Module

When renaming declarations that have references on excluded lines (due to conditional compilation directives) then you will be shown a dialog asking whether you want to rename the 'out of scope' references as well as the 'in scope' references (i.e. to ensure that your Project continues to compile and run if the currently 'out of scope' code becomes 'in scope'). For more on this, see Conditional compilation.

Excel only: when renaming a Sub or Function in a standard Module, any Shape that has an OnAction property that references that Sub or Function will have its OnAction property updated for the new Sub / Function name. In addition, when renaming a standard Module, any Shape that has an OnAction property that explicitly references that Module (as part of the reference to a Sub / Function) will have its OnAction property updated for the new Module name.

Renaming will block certain names (such as same name already used in same scope when VBA rules say that doing so would be invalid; reserved words; for Class Modules that are Interfaces, names that include characters that would prevent its use as an Interface) and warn about others (such as a name that is already used in the same scope and so the new name would be 'hidden').

Renaming supports renaming of all types of declarations including:
- The Project
- Modules (document, standard, Class and UserForm)
    - … including Class Modules that are Interfaces
    - … but not ThisOutlookSession in Outlook
- Enums and Enum members (including Enum member names that require 'square brackets')
- Types and Type elements
- Procedures and Properties
- 'Declare' Procedures (ie those that access the Windows API)
- Events

- Controls on UserForms (and Access Reports and Forms … including handling event handler prefixes, the 'EventProcPrefix')
- Variables (Module-level and Procedure/Property-level)
- Constants
- Line labels

Attributes names for Module-level members (Procedures, Properties, variables and constants) are also updated.

The rename command will assist with renaming declarations when only the case of the declaration needs to be changed. The preferred casing for each type of declaration can be set in the Settings

The Project needs to have been saved at least once for the rename command to work.

Qualify reference

If the cursor is on a reference that is to a Module-level member of another standard Module of the same Project (i.e. a standard Module other than the Module the reference is in), this will qualify the reference with the name of that other standard Module if it is not already qualified e.g. if procedure MyProc is defined in Module1 and is called from a procedure in Module2 as just `MyProc` then this will convert the call to `Module1.MyProc`

If the cursor is on a reference that is to a Module-level member of another standard Module in another referenced-to (ie 'child') Project, this will qualify the reference with the name of that other Project and Module if it is not already qualified.

If the cursor is on an Enum member, this will qualify the Enum member with either the Enum name, the Project name (if relevant) or both e.g. if Enum member THIS_MEMBER is defined in MyEnum and is referenced as just `THIS_MEMBER` then this will convert it to `MyEnum.THIS_MEMBER` (assuming the Enum is declared in the same Project).

The 'Code' tab of the VBE_Extras Settings includes options for the Qualify reference command.

'Qualify reference' is also 'Type Library aware' (and works with Special binding) – for example if you type

```
Dim fso As FileSystemObject
```

… and then place the cursor on `FileSystemObject` and trigger 'Qualify reference' then it will update the line to

```
Dim fso As Scripting.FileSystemObject
```

… assuming, of course, that your Project currently has a reference to the Microsoft Scripting Runtime Type Library.

Add Properties

Auto-add a Property Get, Let and (where appropriate) Set for a Module-level variable or a Type element. Cursor must be on the name of the variable / element within its declaration. Cannot add Properties for variables or Type elements that have a value type of a fixed-length String, another Type, or a static array.

Use the 'Code' tab of the VBE_Extras Settings to set various defaults for the Properties.

To see more background on automatically adding Properties, see my blog post
https://www.thevbahelp.com/post/automatically-adding-properties

Add Factory

Auto-add a Factory and associated Property Get, Let and/or Set declarations for a Class Module. The Factory is designed to populate either the members of a specific Type or the Module-level variables of the Class Module. The Class Module must be the active Module in the VBE. It is not possible to add Properties for variables or Type elements that have a value type of a fixed-length String, another Type, or an array.

Adding a factory also sets the VB_PredeclaredId attribute to True so that the Class has a default instance allowing Class instances to be created using code such as

```
Dim c1 As Class1
Set c1 = Class1.Create("Hello World", 42)
```

Use the 'Code' tab of the VBE_Extras Settings to set various defaults for the Factory and associated Properties.

To see more background on factories, what they are and why to use them, see my blog post https://www.thevbahelp.com/post/automatically-adding-factories

If the name of any of the Property Let/Set/Get(s) added as part of the Factory would clash with an existing member or clash with a VBA keyword, then a numeric suffix will be added e.g. if you add a Factory for a Type and that Type includes an element 'sglNew As Single' and you have the Setting for 'New property prefix behaviour' set to 'Remove entire prefix', then the new Property will be named 'New1' (and not 'New' which would clash with the VBA keyword of the same name).

Extract to constant

Extract literal values to a constant. The cursor must be on a literal value or the name of a constant in the Const declaration. Excludes literal values used in conditional compilation (either as the value assigned to a #Const or in an #If or #ElseIf statement).

If the cursor is on a literal value other than in the Const declaration

A new constant will be created and all literal values matching that at the cursor will be updated to use the constant. You can define the scope the literal values are extracted from, where the Const declaration will be put, the name and value type of the Const declaration and whether literal values within code comments are extracted or not. If the literal value is a String then matching can be case sensitive or not (the new constant will use the case of the String at the cursor).

Literal values that are the default values of optional Procedure / Property parameters are Module-level (not Procedure / Property-level) and will only be extracted to a Const if the scope to extract from is Module or Project (not if the scope is Procedure / Property).

VBE_Extras will first check for any existing Consts anywhere in the Project that define the same value. If any are found, a dialog will be shown.

If the cursor is on the name of a constant in the Const declaration or on the literal value being assigned to that Const

VBE_Extras will search for other matching literal values and replace them with the name of the Const. If Const is a String then matching to other literal values can be case sensitive or not.

Only the scope of the Const will be used for the search for other literal values … if the Const is declared:

- Public in a Module then the scope will include other literal values within all Modules of the VB Project
- Private (or default) in a Module then the scope will include other literal values within that Module
- In a Procedure / Property then the scope will only include other literal values within the same Procedure / Property (not including default values for optional parameters)

Note that this command only works with a Const that uses a single literal value and not with a Const that uses an expression that needs evaluating (eg `Const MY_NUM As Long = 21 + 21`)

General

Some literal values will not be extracted to a Const: those that are already the value used in a Const or #Const declaration; those used in a conditional compilation directive line (eg #If, #ElseIf); those used in a Declare Sub / Function line.

If 'Match conditional compilation directives' is selected in the Settings then literal values on excluded lines will not be extracted.

To see a list of literal values, see Matches for literal value at cursor but note that command can display ALL matching literal values whereas Extract to constant is sensitive to code scoping rules.

To understand in more detail how a 'match' between literal values is determined, see Matches for literal value at cursor.

Extract to constant is single-Project and will only extract values from within the same Project.

Implement Interface

Only Class Modules and UserForm Modules can implement an Interface.

To be considered an Interface, a Module must:
- Be a Class Module
- Not implement another Interface
- Have one or more Public members - either a Property, a Sub or a Function with an empty body, or a field
- None of the Public members must have an underscore in their name
- Not include any conditional compilation [1]
- Be in the same Project as the Class or UserForm Module that it is to be implemented in, or be in a 'child' Project [2]
- If the host application is Access, the Interface name must not include characters (e.g. a space character) that would require the use of square brackets [3]
- If the host application is Access and the Interface is in a 'child' Project, the 'child' Project name must not include characters that would require the use of square brackets [3]

[1] this is a requirement of VBE_Extras, not a requirement of VBA
[2] if in a 'child' Project, the Interface must have its VB_Exposed attribute set to True … VBE_Extras will set this when implementing the Interface
[3] as the Implements keyword does not allow the use of square brackets

If a Module could be considered to be an Interface but is already implemented by the target Module (both an Implements statement and each required member is present†) then VBE_Extras will not include the Interface in the list of those that it can implement for a particular Module ... however, if the Implements statement is missing or any one (or more) of the members is missing then VE will include the Interface in the list of those that it can implement and will add only the relevant code (Implements statement and/or members).

† only the presence of a Procedure / Property with the right name and unit type is checked ... the parameters and (if relevant) returned value type are not checked

As well as the required Interface members, VBE_Extras will offer to add to the implementing Class / UserForm:
- Object members - so that the Interface members are accessible when client code is working with instances of the implementing Class / UserForm
- Fields for implemented Properties (along with code to set/get the field's value) - if no clashing members already exist in the implementing Class / UserForm

If the implemented Interface is in a 'child' Project then the Interface Class Module will have its VB_Exposed attribute set to True (if it is not already) to comply with VBA rules.

Extract Interface

Create a new Interface from one or more members of a Class or UserForm Module. The Class or UserForm Module must have one or more Properties or Procedures that:
- Do not have an underscore in their name
- Do not have a value type of a Type, fixed-length String or array (static or dynamic)
- Do not have a value type of an Enum that is defined in the same Class or UserForm Module that the interface is to be extracted from

A new Interface (a Class Module) will be created and that newly created Interface will be implemented in the Class or UserForm Module – the added Interface members can either be empty or can call to the (existing) members that were used to create the Interface.

The new Interface can be added to the same Project or to a 'child' Project (so long as that child Project has a name that does not require the use of square brackets ... as the VBA 'Implements' keyword does not support square brackets). If the Interface is added to a child Project then the (newly created) Class Module that is the Interface will have its VB_Exposed attribute set to True.

Make 'Option Explicit'

Only available when the Office Display Language is set to be English.

For a Module that does not currently have an Option Explicit statement, adds that statement and then adds explicit 'Dim' declarations for all undeclared variables. Optionally, will create a report of all variables added and where they were added.

IMPORTANT:
- You must ensure your Project has no other compilation errors (i.e. in the VBE menu, Debug > Compile does not result in an error dialog) before using this command
- If you need to exit the Make 'Option Explicit' process, hold down the Alt key
- Do not use the keyboard (other than the Alt key to exit, if required) or click with the mouse while this command is executing ... doing so will interfere with the process and likely cause the process to end before completion (however, you can just start the process again)
- No progress bar is shown while the process is running (a progress bar would interfere with the process and so cannot be shown)
- Your PC may repeatedly 'ding' while the process is running … this is normal and cannot be prevented (turn down your volume if required)
- Pauses of a few seconds can occur while this command is running … this is normal … when the process is complete, a dialog will be displayed

Note that:
- Make 'Option Explicit' does not take conditional compilation into account ... you will have to manually check that the declarations added cover all conditional compilation scenarios
- When the same variable appears in multiple procedures / properties, you will be asked whether you want to add one declaration at the Module-level or in each procedure / property
- All variables are added 'As Variant'

Update parameters

Update parameters for the procedure or Property at the cursor and simultaneously update the arguments for all references of that procedure or Property.

Each 'Update parameters' command:
- Correctly handles matching Properties (i.e. any matching Get / Let / Set will also be updated)
- Correctly handles both positional and named arguments
- Correctly handles required, Optional and ParamArray parameters including (for Optional and ParamArray) omitted arguments and arguments that are provided as an 'empty comma'
- Is 'multi-project' (see Referenced VBA Projects for more)
- Will preserve any existing Attributes

Excluded from scope:
- Procedures and Properties that are Interface members and their implementations
- Events and event handlers
- Procedures and Properties that are called using a String e.g. 'Application.Run' or the 'OnAction' Property of a CommandBarButton

Conditional compilation:
- When a procedure or Property has multiple declarations due to conditional compilation, only the compiled-in declaration will be updated (a warning will be shown when this is the case)
- All references of a procedure or Property will be updated whether they are compiled-in or not

Add parameter

Add a parameter to a procedure or Property and simultaneously add an argument for all references of that procedure or Property.

- For the arguments added for the references of the procedure or Property – normally positional arguments will be used, named arguments will be used if a reference already uses one or more named arguments AND if the new argument comes after that named argument

- Line breaks within the parameters (and arguments at the call site of the procedure / Property) will be preserved ...
    - If there are 3 or more parameters in the procedure / Property signature and each is on a separate line then additional parameters will also be added on their own line
    - If there are 3 or more arguments at the call site of the procedure / Property and each is on a separate line then additional arguments will also be added on their own line
- If the host application is Excel:
    - If the procedure is 'assigned as a macro' to a Shape then adding a 'required' parameter will break the link from the Shape ... a warning will be shown if this is the case
    - If the procedure is the target of a keyboard shortcut then adding either a 'required' or an Optional parameter will break the keyboard shortcut ... a warning will be shown if this is the case
- In the 'Add Parameter' dialog, only very limited validation is carried out for the 'Default value' for Optional parameters and for the 'Value to inject at call site(s)' ... and the validation that is carried out for both of these can always be overridden

Promote local variable to parameter

Promote a local variable within a procedure or Property to be a parameter of that same procedure or Property and simultaneously add an argument for all references of that procedure or Property.

- The cursor can either be on the procedure / Property declaration line or on the name of the variable that is to be promoted
- Only variables can be promoted: not Statics, Consts
- If the variable is an array, it must be 'dynamic' not 'static' (i.e. it must not be defined with explicit bounds)
- Any code comments on the line of the variable that is to be promoted will be preserved
- The relevant comments in Add parameter also apply when 'promoting a local variable'

Remove parameter

Remove a parameter from a procedure or Property and simultaneously remove the matching argument from all references of that procedure or Property.

Demote parameter to local variable

Remove a parameter from a procedure or Property and make it a local variable within that same procedure or Property and simultaneously remove the matching argument from all references of that procedure or Property.

- Demoting always creates a Dim statement (never a Static or Const)

Reorder parameters

Reorder one or more parameters of a procedure or Property and simultaneously reorder the matching arguments of all references of that procedure or Property.

- A ParamArray parameter cannot be reordered (it must be the last parameter)
- The value/reference parameter of a Property Let/Set cannot be reordered (it must be the last parameter)
- Parameters can be updated from being required to Optional, and vice-versa, depending on the parameter order
- If an Optional parameter is being reordered to an earlier position in the list of parameters and is being made 'required' and if that parameter is currently omitted at any reference of the procedure/Property then a 'Value to inject at call site(s)' will be required (i.e. to use as the required argument for the now-required parameter)

Tasks in this Project

Displays a dialog with a list of all tasks (aka TODOs) in the active Project. Tasks are always in code comments that use an apostrophe (not a Rem statement).

The specific text that defines a 'task' can be customised, along with the 'priority' level in the VBE_Extras Settings. The specific text can immediately follow the code-comment apostrophe or have one or more space characters in-between. Any other character in-between will prevent the task from being identified.

Goto

Go to declaration for code at cursor

Just as it sounds, if the cursor is on a reference, then the cursor will be moved to that reference's declaration. Very similar to pressing the Shift + F2 keys (which activates the VBE's built-in 'go to declaration' command), but this works for more types of declaration and is more accurate.

There are limitations when the declaration is a Control in an Access Form or Report … see Using VBE Extras in Access.

Go to a declaration in this Module

Displays a dialog with a list of all declarations in this Module, allows you to select one to 'go to'.

Go to line in this Module

Enter a line number, go to it. Optionally follow the line number by a comma then a column number to go to a column within the line.

Go to Module

Display all Modules in the active Project, allows you to select one Module to 'go to'.

Note that Class Modules that are Interfaces are distinguished using a separate icon from the normal Class Module icon.

If 'Indicate VB_PredecaredId' is checked in the UI tab of the Settings dialog then the names of Class Modules that have the VB_PredeclaredId attribute set to True will be followed by 'PdI'.

If 'Indicate VB_Exposed' is checked in the UI tab of the Settings dialog then the names of Class and UserForm Modules that have the VB_Exposed attribute set to True will be followed by 'Exp'.

Go to last edit

Go to the location that code was last edited.

Note that:

- Edits in Projects that have never been saved are not recorded
- Only edits to code are recorded and not, for example, changes to UserForm layout, Modules being added / removed
- Updates to Attributes are not recorded
- For those commands that operate on an entire 'scope' (procedure, Module or Project), for example adding or removing line numbers, the 'last edit location' is set to the location of the cursor at the time of the operation
- For those commands that (potentially) replace multiple Modules in a Project, for example Version Control (Import), Replace all Modules and Clean this Project, the 'last edit location' will be cleared
- Edits made using 3rd party VBE AddIns may not be recorded

Go to implementation

For a member of an Interface (i.e. a Class Module being used as an Interface), go to an implementation of that member.

For the Class Module itself, if clicked in the Project window (where that Class Module is an Interface), go to a Class that implements the Interface.

Go to base

For a member of a Class or UserForm Module (where that Module implements an Interface), go to the base for that member in the Interface.

For the Class or UserForm Module itself, if clicked in the Project window (where that Module implements an Interface), go to the Interface.

Go to previous code reference in this Project / Go to next code reference in this Project

Go to the previous or next code reference of a declaration for the code at the cursor.

This excludes references in code comments, in Strings or (for Excel) the OnAction property of Shapes. These commands are single-Project only (will not go to references in other Projects).

Go to declaration above / Go to declaration below

Go to the Module-level declaration above or below the current cursor position within the Module currently displayed.

Expand selection / Contract selection

Expand or contract the code selection … depending on what is selected when you first 'expand', cycles from:

- An insertion point
- A word
- A physical line
- A logical line
- A block of code … expanding through wider blocks within a procedure *
- A procedure
- A procedure plus preceding blank and/or comment lines
- … and cycles back when you 'contract'

If the initial selection is within code comments, cycle is:

- An insertion point
- The code comments (on one line)
- All consecutive lines with code comments
- A block of code … expanding through wider blocks within a procedure *
- A procedure
- A procedure plus preceding blank and/or comment lines
- … and cycles back when you 'contract'

* If the selected line is within an Enum or Type in the Module's declaration lines, then the Enum or Type will be selected

Conditional compilation lines are ignored.

Go to Bookmark in this Project

View the bookmarks in the active Project, optionally 'go to' one.

Go to Shape with this name or that calls this Procedure

Only available in Excel.

The Project must have been saved at least once for this command to work. This command is single-Project and will only find Shapes in the Workbook that is associated with the active Project.

If the cursor is on the name of a Sub or Function in a standard Module

For a Sub or Function in a standard Module, if it is the target of one or more Shapes' OnAction property (as set using 'Assign Macro') then go to that Shape in the relevant Worksheet or Chart sheet.

For the reverse action, see Go to OnAction

If the cursor is on a String literal or Const

For the String literal or Const (with a String value-type, or a Variant value-type holding a String), go to the Shape with that name in the relevant Workbook or AddIn:

- If the VBA Project containing the String literal / Const is in a Workbook, only that Workbook is searched for the Shape
- If the VBA Project containing the String literal / Const is in an AddIn:
    - The AddIn itself will be searched first (even though an AddIn has no user-interface, it can still contain Shapes)
    - If there is an active Workbook (in the Excel user-interface) then search that second

Note this command is not 'code-context sensitive' i.e. it does not attempt to parse the code to determine which Worksheet or Chart sheet (within the Workbook or AddIn) the Shape is in … or even if the String literal / Const is referring to a Shape at all, rather if multiple Shapes with the same name exist (ie in multiple Worksheets / Chart sheets) then they will all be listed for you to select one.

For the reverse action, see Uses of Shape's name

Go to Table with this name

Only available in Excel.

The same as Go to Shape with this name or that calls this Procedure when the cursor is on a String literal or Const but for Tables (and note that Table names are unique within a Workbook so you will never be shown a list of Tables to select one).

For the reverse action, see Uses of Table's name

Go to Name with this name

Only available in Excel.

The same as Go to Shape with this name or that calls this Procedure when the cursor is on a String literal or Const but for Names that refer to a (fixed) Range.

For the reverse action, see Uses of Name's name

Sub to run

Not available in Outlook.

The Project needs to have been saved at least once for the 'Sub to run' commands to work.

Note that when Word is the host application, you may see the screen flickering when you 'Run Sub to run' or 'Run Again' … this is because Word requires the Document that contains the Sub being run to be the active Document and so VBE_Extras makes the Document active and then re-activates the VBE.

Despite the name, all of these commands work on VBA Functions, as well as VBA Subs. However, they must be in standard or document Modules and must have no parameters.

Set Sub to run / Run Sub to run / Clear Sub to run

Want to edit code in one place in your Project and quickly be able to run the code from another point … these are the options for you.

If you press the keyboard shortcut for 'Run Sub to run' and no Sub is set, VBE_Extras will assist you to select a relevant Sub.

'Run Sub to run' also gives you the option to either jump to the end of, or clear, the Immediate window before the Sub is run (set this option in the 'Run' tab in the Settings).

Run Again

The 'Run Again' command can be enabled / disabled in the 'Run' tab in the Settings.

When 'Run Again' is enabled, VBE_Extras records the Sub you last ran via F5 or via the Run > Run Sub/UserForm menu item. Then, when you select to 'Run Again', that Sub is then run. Meaning you don't have to keep navigating around trying to find the Sub in order to run it … and you don't have to navigate around trying to find the code you were editing afterwards.

'Run Again' also gives you the option to either jump to the end of, or clear, the Immediate window before the Sub is run (set this option in the 'Run' tab in the Settings).

Note that:

- VBE_Extras only records the Sub you last ran via F5 or via the Run > Run Sub/UserForm menu item and not when code is run via any other method e.g. not when code is run via event handlers, RibbonX callbacks, F8, or clicking a Shape, or via Application.Run or via the 'Macros' dialog or via VBE_Extras 'Run Sub to run' command … or any thing else other than F5 or the Run > Run Sub/UserForm menu item
- If you use Run > Run Sub/UserForm to 'run a UserForm' then the Sub to 'Run Again' will be cleared
- 'Run Again' only works for code that is run within Projects that have been saved at least once

Immediate window: clear / go to start / go to end

Yep, do these things to the Immediate window

The Project needs to have been saved at least once for the 'Immediate window' commands to work.

Also, you can configure (via an option in the 'Run' tab in the Settings) VBE_Extras to automatically clear or go to the end of the Immediate window when a Sub (or Function) is run using:

- F5 or the Run > Run Sub/UserForm menu item
- F8 or the Debug > Step Into menu item

Lines

Add / update line numbers

Add / update line numbers – to either the Project, a whole Module or to a specific Procedure / Property.

If existing line numbers are 'referenced' (e.g. in GoTo, GoSub or Resume statements) then references will be maintained when updating line numbers.

Options for the first line number, the step to subsequent line numbers, and whether line numbering is reset at the Module level or at the Procedure / Property level are available in the 'Other' tab of the Settings.

Preserves Attributes if present.

Remove line numbers

Remove line numbers – from either the Project, a whole Module or from a specific Procedure / Property.

If existing line numbers are 'referenced' (e.g. in GoTo, GoSub or Resume statements) then those line numbers will not be removed in order to keep code 'compile-able'. If you wish to remove 'referenced' line numbers, you should convert them to line labels.

Preserves Attributes if present.

Indent lines

Apply code indentation rules – to either the Project, a whole Module, the Module declaration lines or to the current Procedure / Property.

Handles line numbers if present. When line numbers are present, it can appear that too much whitespace is left between the line number and the indented code, however, this is because the 'End' line of a Procedure or Property CAN have a line number (though typically doesn't … even when line numbering is applied) and space is left for it even if it is not present.

Preserves Attributes if present.

There are options relating to indenting in the 'Indent' tab of the Settings:

- Size of indent (in characters)
- Whether to indent within conditional compilation blocks
- Whether to indent within On Error blocks
- Whether to apply indentation rules to lines before the first code line of a Procedure / Property
- The size of indent for continued lines

Note that line labels are required to always start in column 1 (i.e. to have no indent).

Oddities … while VBE_Extras does indent the following types of lines correctly, by the time the line is displayed to you by the VBE, the indentation has been changed (by the VBE … try to indent them manually and see!):

- Lines that start with a colon are always forced to column 1 by the VBE
- Lines that consist only of the line continuation characters (i.e. " _") are always forced to column 1 by the VBE

Split lines

Split the logical line(s) at the cursor into individual physical lines (using the line continuation characters sequence " _"). Additionally, if a line is already split (but at a point that doesn't align with your settings) then the line with be joined.

Use the Settings to set your preferences for:

- The maximum physical line length
- Whether a logical line that is split into multiple statements using ':' should be broken down into discrete logical lines for each of those statements before the 'split' takes place or whether the multiple statements should be treated as one long logical line
- Whether a '.' and a '&' (to split Strings) should be positioned at the end of the (previous) physical line or at the start of the (next) physical line

These settings are also applied when VBE_Extras automatically adds a line of code to your project, for example when using:

- The Add Factory command - the Create() Function declaration can have a lengthy line if it has many parameters, and
- The Make 'Option Explicit' command - if many declarations are added on the same line

Comment / Uncomment lines

Comment / uncomment the line(s) at the cursor. If 'Comment / Uncomment expands' is selected in the Settings then the lines commented / uncommented will be expanded to the start and to the end of a logical line of code (i.e. where line continuations are involved).

Note that commenting line(s) of code that have Attributes will remove those attributes. The VBE itself does not allow commented lines to have attributes.

Move lines up / down

Move the selected line(s) up or down by one line. The detail of how this command behaves is determined by the 'Move lines up/down expands' setting in the Settings:

If checked: the line(s) being moved up / down and the line(s) that are being 'moved over' are expanded to be 'logical lines' (that is, a line with one or more line continuation(s) is treated as a single line). Additionally, if the logical line(s) have line numbers then they will be maintained in the pre-existing order (i.e. the code on the lines moves up / down but the line number on the lines does not). Attributes are always preserved.

If not checked: the line(s) are moved based on physical lines (that is, line continuations are ignored). Additionally, if the physical line(s) have line numbers then they are treated as being part of the line and are moved with it (i.e. they are not updated). Note

this does not apply any 'code sense' to the selected line(s) … it will move just the selected line(s) whether they are lines ending with continuation characters (or are being moved into other lines ending with continuation characters). Attributes are preserved except for where line continuations are broken as a result of the operation. It is not possible to preserve Attributes for 'broken' lines.

Other tools

Attributes

For more on VBA attributes, what they do, how to use them, and examples, see the 'VBA attributes' blog post at https://www.thevbahelp.com/post/vba-attributes

Note that the Main menu and the Code window context menu have options to view/set all attributes (both Module attributes and Procedures, Properties, variable and constant attributes); the Project window context menu has options to view/set Module attributes only.

All VBE_Extras commands that update your VBA code will preserve existing Module attributes. The majority of VBE_Extras commands that update your VBA code will preserve existing Procedures, Properties, variable and constant attributes, for example:

- Rename code at cursor – if the member being renamed has an attribute, it will be retained (and updated to match the new name of the member)
- Move lines up / down – if the line that is being moved up or down has an attribute, it will be moved with the line; if the line being 'moved over' has an attribute then it will be retained

However, some VBE_Extras commands can't preserve any existing attributes (e.g. Comment / Uncomment lines … in this case because the VBE does not allow attributes on commented-out lines) and you will be warned, when an attribute is present that will be removed, before the command is performed.

Procedures, Properties, variable and constant attributes

View attributes

View all Procedure, Property, variable and constant attributes in the active Module.

Set 'VB_Description' attribute

Set the value of the VB_Description / VB_VarDescription attribute for a Procedure, Property or Module-level variable or Module-level constant in a standard, Class or UserForm Module. You must right-click on the specific member name (i.e. not within the declaration or body of the member). Additionally: for an event handler, you must right-click on the event part of the name (not the name of the Object that omits the event) to the right of the underscore; for a variable or constant, the variable or constant must be declared on its own line (i.e. not multiple on a single line delimited by a comma or colon).

Once applied, a description can be seen in the VBE's Object Browser window.

VBE_Extras cannot set a description for members of Document Modules (other than Forms and Reports in Access) as Document Modules cannot be re-imported following the attribute being added / changed.

All descriptions have a max length of c 900 characters; speech marks are automatically 'escaped' for you.

Set 'VB_UserMemId = 0' Attribute

Set the value of the VB_UserMemId / VB_VarUserMemId attribute to 0 for a Procedure, Property or Module-level variable or Module-level constant in a Class Module. You must right-click on the specific member name (i.e. not within the declaration or body of the member). Additionally, for a variable or constant, the variable or constant must be declared on its own line (ie not multiple on a single line delimited by a comma or colon).

Only one member in any one Class Module can have this attribute set with a value of 0. This attribute makes that member the default member of the class. Note also that this attribute is mutually exclusive with the 'VB_UserMemId = -4' attribute (i.e. any one member cannot be both the default member and return an enumerator).

Set 'VB_UserMemId = -4' Attribute

Set the value of the VB_UserMemId attribute to -4 for a Function or Property Get in a Class Module. You must right-click on the specific member name (i.e. not within the declaration or body of the member).

Only one member in any one Class Module can have this attribute set. This attribute makes that member return an enumerator such that a Collection member of the class can be enumerated using a 'For Each' loop. Note also that this attribute is mutually exclusive with the 'VB_UserMemId = 0' attribute (i.e. any one member cannot be both the default member and return an enumerator).

Module attributes

View attributes

View the attributes in the Module header for a Module of any type (document, standard, Class or UserForm).

Set 'VB_Description' attribute

Set the value of the VB_Description attribute in the Module header for a standard, Class or UserForm Module.

Once applied, a description can be seen in the VBE's Object Browser window.

VBE_Extras cannot set a description for Document Modules (other than Forms and Reports in Access) as Document Modules cannot be re-imported following the attribute being added / changed.

All descriptions have a max length of c 900 characters; speech marks are automatically 'escaped' for you.

Set 'VB_PredeclaredId' attribute

Set the value (to True or False) of the VB_PredeclaredId attribute in the Module header for a Class Module. The default for a newly added Class Module is always False. Setting this to True gives the Class Module a default instance.

If you want to view the VB_PredeclaredId state for all Modules in a Project, you can use the 'Go to Module' dialog (without actually 'going to' a Module). See Go to Module for more details.

Set 'VB_Exposed' attribute

Set the value (to True or False) of the VB_Exposed attribute in the Module header for a UserForm (or Class Module … though this can be adjusted anyway for a Class Module in the VBE Properties window). The default for a newly added Module (either UserForm or Class) is False. Setting this to True makes new instances of the Module public (though it does not make it externally 'creatable').

If you want to view the VB_Exposed state for all Modules in a Project, you can use the 'Go to Module' dialog (without actually 'going to' a Module). See Go to Module for more details.

Matches for literal value at cursor

If the cursor is on a literal value (or on the name of a constant in a Const or #Const declaration) then this will display a dialog with a list of all matching literal values in the active Project. The dialog will allow you to set the scope of values shown (Project, Module or Procedure/Property – based on the location of the literal value at the cursor), to show or hide literals that are in code comments and, for String literals, whether matches must be case sensitive or not.

If 'Match conditional compilation directives' is selected in the Settings then literal values on excluded lines will not be shown.

How is a 'match' determined?

All literals are broken down into 4 'literal types': Strings (including fixed-length), numerics (all numeric types), Dates and Booleans.

Matches do not happen *across* 'literal types' e.g. a String "42" will not match to a numeric 42.

Matching does not distinguish *within* 'literal types' e.g. a numeric 42 assigned to a Long is the same as 42 assigned to a Double.

When matching numeric 'literal types', type-declaration characters are ignored e.g. 42& is the same as 42 … however, different bases do not match e.g. &H2A is not the same as 42.

Orphan declarations

The 'orphan' commands identify unused code.

In this Procedure / Property / Module / Project

Identifies orphan declarations within the selected scope (i.e. procedure/property, Module or Project).

A variable (procedure / property-level and Module-level) will be identified as an orphan if it has no references, or if all references only write to it (i.e. never read from it), or if all references only read from it (i.e. never write to it). In addition:

- A variable passed to another procedure / property 'ByVal' is a 'read-only' reference and the variable will be listed as an orphan if it is not written-to elsewhere in the owning procedure / property
- A variable being passed to another procedure / property 'ByRef' (which is the default if not specified) is a 'read/write' reference and the variable not be listed as an orphan

An Event will be identified as an orphan if it has no references to it, or if all references only Raise the Event, or if all references only handle the Event.

A Control will be identified as an orphan if no references to it exist (references include direct references of the Control object or Event handlers for the Control). As certain types of Control (notably Labels, Frames and Images) often, though not always, do not have direct references or Event handlers (but are still a valid part of a UI layout) then they can be excluded from the list of displayed orphans, using the drop-down (the drop-down is only shown when Controls are present in the list of orphans)

The orphan commands will not report a Sub or Function to be an orphan if:

- It is an Event handler (e.g. for ActiveX controls or Events defined in Class Modules … but see Detached event handlers)
- In Excel, it is referenced in the OnAction property of a Shape
- In Excel, it is the target of a keyboard shortcut set via the 'Macro Options' (Alt+F8) dialog or via the Application.MacroOptions() method

Declarations will be considered to be an orphan if they are only referenced by:

1. Code in other Projects and those other Projects are not open or the 'Multi-Project' option in Settings is not checked
2. In Word, MacroButton Fields or FormFields

Note that:

- VBE_Extras does not read Ribbon XML and so callbacks from Ribbon XML, by default, are considered to be orphans … however, you can change this using the 'Ribbon XML callbacks are not orphans' option in the Settings such that a Sub in a standard Module that takes a parameter of type IRibbonUI or IRibbonControl is excluded from the orphans list
- Strings are not normally considered to be 'code' references and so a string with the same name as a declaration (with the parent Module and/or Project as appropriate) would not normally stop a procedure declaration from being considered to be an orphan, however, you can change this using the 'Procedures called by strings are not orphans' option in the Settings such that a Sub or Function in a standard or document Module that is referenced from a single string literal or a single Const in a call to, for example, Application.Run or Application.OnTime is excluded from the orphans list … note the reference must

be from a single string literal or a single Const without any form of manipulation (eg concatenating multiple string literals or Consts; using a variable)
- To stop a specific procedure from being flagged as being an orphan, you can Insert '@EntryPoint'

Access only: see Using VBE Extras in Access

Orphan Modules

Checks whether Standard, Class and UserForm Modules are unused. Document Modules (e.g. ThisWorkbook and Sheet Modules in Excel; Form and Report Modules in Access) are never orphans as they are fundamental elements of the parent file (e.g. Workbook in Excel; Database in Access).

A Standard Module is considered to be unused if none of the members it contains are referenced.

A Class or UserForm Module is considered to be unused if it is never referenced (e.g. in a Dim or With statement).

Declarations that could be made Private

That is, find declarations that are only referenced from within their own Module

Note that this command only looks in the VBA code for references. To prevent a Procedure from being incorrectly flagged as being one that could be made Private, you can Insert '@EntryPoint'

Detached event handlers in this Project

That is, event handlers that handle events for objects that no longer exist. For example, if you have a UserForm with a 'CommandButton1' control then you might have an event handler 'CommandButton1_Click' to handle it's click event. If 'CommandButton1' is then deleted or renamed but 'CommandButton1_Click' is not, then you have a detached event handler. This command will find that detached event handler.

This command is single-Project only (it will find only detached event handlers in the active Project).

Keyboard shortcuts in this Project

Only available in Excel.

Shows keyboard shortcuts applied with the 'VB_ProcData.VB_Invoke_Func' attribute - that is, those applied using the 'Macro Options' (Alt+F8) dialog and those applied using the Application.MacroOptions() method ... specifically, it does not show keyboard shortcuts applied programmatically using the Application.OnKey() method.

Call hierarchy

View 'calls of' and 'calls from' the procedure or Property at the cursor. Once the lists of 'calls of' and 'calls from' are shown, you can drill-down into further levels of 'calls of' and 'calls from' allowing you to navigate all calls to and from a selected procedure or Property.

Note that:
- The 'calls' are static design-time calls … this command does not provide a view of the run-time call stack
- The previously viewed 'call' will be shown in bold … use the 'Back' button to quickly go back to this 'call'
- If any 1 or more calls are recursive then the "Recursive" column will be shown indicating which calls are recursive
- While Call hierarchy is multi-Project, it works for only 1 generation of Project parent/child relationship either way:
    - It will find calls of the initial procedure or Property (that Call hierarchy is shown for) in parent Projects but if you then jump to a procedure or Property in a parent Project then it will not find calls of that procedure or Property in its parent Projects (i.e. to grand-parent Projects of the initial Project)
    - It will find calls from the initial procedure or Property (that Call hierarchy is shown for) to child Projects but if you then jump to a procedure or Property in a child Project then it will not find calls from that procedure or Property to its child Projects (i.e. to grand-child Projects of the initial Project)

Fix Case

Change the case of the text at the cursor. The text can be any code (not a literal value or a comment) other than a VBA keyword (the casing of which cannot be changed). The purpose of this command is to fix incorrect casing relating to the VBE's habit of changing case throughout a Project based on the name of any declaration.

For example, if you add a declaration "value" to your Project, then all instances of "Value" (or "VALUE" or any other case of the same word) throughout your Project will be adjusted to "value" … for example, in Excel, the Value property of a Worksheet cell.

This function allows you to fix the casing without needing to find the declaration that caused the case to change … just put the cursor on any instance of the word "value", trigger the Fix Case command and then enter the correct casing (in this case, "Value").

Note that ALL instances of the same word will be changed case throughout your Project … there is no way to avoid the VBE's habit of changing case throughout a Project for all items with the same name.

Insert '@EntryPoint'

Supports the functionality of the Orphan declarations commands and the Declarations that could be made Private command. To prevent a Procedure from being incorrectly flagged as being an orphan or one that could be made Private, right-click anywhere within that Procedure to insert the '@EntryPoint' text. This text must be inserted between the Procedure's body line and the 'End' line.

Add Enum from Type Library

Replicate an Enum from a Type Library into your VBA code. Handy if you develop code with early binding but will release your code with (or otherwise will subsequently change to using) late binding.

For example, if you are automating Outlook from another host application and you struggle to remember which OlItemType refers to which type of Item then use 'Add Enum from Type Library', select the 'Outlook' Type Library then select the 'OlItemType' Enum and this will be added to your VBA code …

```
Enum OlItemType
    olMailItem = 0
    olAppointmentItem = 1
    olContactItem = 2
    olTaskItem = 3
    olJournalItem = 4
    olNoteItem = 5
    olPostItem = 6
    olDistributionListItem = 7
    olMobileItemSMS = 11
    olMobileItemMMS = 12
End Enum
```

Add a "GetEnumAsString()" Function

If you define a variable (or other declaration) as having the value type of an Enum and then want to get the value (i.e. the Enum member) that has been assigned to it, you will just get the numerical value – there is no simple way to get the name of the Enum member. In other words, if you have this Enum …

```
Enum Fruit
    FRUIT_APPLE
    FRUIT_ORANGE
    FRUIT_PEAR
End Enum
```

… and then run this code …

```
Dim f As Fruit
f = FRUIT_ORANGE
Debug.Print f
```

… then "1" will be sent to the Immediate window (as the default Enum member numbering is 0-based) which isn't ideal unless you can remember the exact order of each Enum member. What if you wanted the actual name of the Enum member assigned to the variable (i.e. "FRUIT_ORANGE" in this case)? You would need something like this:

```
Function GetFruitAsString(value As Fruit) As String
    Dim res As String
    Select Case value
        Case Fruit.FRUIT_APPLE: res = "FRUIT_APPLE"
        Case Fruit.FRUIT_ORANGE: res = "FRUIT_ORANGE"
        Case Fruit.FRUIT_PEAR: res = "FRUIT_PEAR"
        Case Else: res = ""
    End Select
    GetFruitAsString = res
End Function
```

… and then change the Debug.Print line to …

```
Debug.Print GetFruitAsString(f)
```

… then "FRUIT_ORANGE" will be sent to the Immediate window which is much clearer as to its meaning. As typing out the equivalent of GetFruitAsString() is time consuming, then use the 'Add a "GetEnumAsString()" Function' to add it for you.

The 'Add a "GetEnumAsString()" Function' works not only for Enums defined in your VBA code but also for Enums defined in Type Libraries that, either, the VBA Project has a Project Reference to or that are 'referenced' using Special binding.

Structure of the GetEnumAsString() Function

Some Enums are designed so that multiple Enum members can be combined (sometimes called 'composite' Enums). An example is VbFileAttribute which can combine a series of file attributes that can be applied to a file, for example a file can be both vbReadOnly and vbHidden. Such Enums have members with values that are 'bit flags' i.e. they are all powers of 2 (optionally with an Enum member with a value of 0 … in the case of VbFileAttribute, that is vbNormal).

The 'Add a "GetEnumAsString()" Function' recognises the 'composite' nature of the Enum and adds a Function with a different structure so that *all* Enum members (that are passed into the 'value' parameter) are returned.

An example of an Enum that is not 'composite' is VbDayOfWeek (its Enum members have sequential values, from 0 to 7) for which 'Add a "GetEnumAsString()" Function' will add:

```
Function GetVbDayOfWeekAsString(value As VbDayOfWeek) As String
    Dim res As String
    Select Case value
        Case VbDayOfWeek.vbUseSystemDayOfWeek: res = "vbUseSystemDayOfWeek" ' 0
        Case VbDayOfWeek.vbSunday: res = "vbSunday" ' 1
        Case VbDayOfWeek.vbMonday: res = "vbMonday" ' 2
        Case VbDayOfWeek.vbTuesday: res = "vbTuesday" ' 3
        Case VbDayOfWeek.vbWednesday: res = "vbWednesday" ' 4
        Case VbDayOfWeek.vbThursday: res = "vbThursday" ' 5
        Case VbDayOfWeek.vbFriday: res = "vbFriday" ' 6
        Case VbDayOfWeek.vbSaturday: res = "vbSaturday" ' 7
        Case Else: res = ""
    End Select
    GetVbDayOfWeekAsString = res
End Function
```

… and the returned String will be the name of the one Enum member that is assigned to your variable and passed in as the 'value' parameter.

An example of an Enum that is 'composite' is VbFileAttribute (its Enum members all have values that are powers of 2, plus one that has a value of 0) for which 'Add a "GetEnumAsString()" Function' will add:

```
Function GetVbFileAttributeAsString(value As VbFileAttribute) As String
    Dim res As String
    If (value And VbFileAttribute.vbReadOnly) Then res = res & "vbReadOnly + " ' 1
    If (value And VbFileAttribute.vbHidden) Then res = res & "vbHidden + " ' 2
    If (value And VbFileAttribute.vbSystem) Then res = res & "vbSystem + " ' 4
    If (value And VbFileAttribute.vbVolume) Then res = res & "vbVolume + " ' 8
    If (value And VbFileAttribute.vbDirectory) Then res = res & "vbDirectory + " ' 16
    If (value And VbFileAttribute.vbArchive) Then res = res & "vbArchive + " ' 32
```

```
    If (value And VbFileAttribute.vbAlias) Then res = res & "vbAlias + " ' 64
    If Len(res) = 0 Then
        res = "vbNormal"
    Else
        res = Left$(res, Len(res) - 3)
    End If
    GetVbFileAttributeAsString = res
End Function
```

… and the returned String will be the names of all of the Enum members that are assigned to your variable and passed in as the 'value' parameter.

Note that:

- An Enum will be identified as being 'composite' if it includes 3 or more Enum members with non-0 values that are all powers of 2 and no Enum members with values that are not powers of 2 (or are not 0).
- Some Enums are 'partially composite', for example VbVarType has values that are mostly sequential (e.g. vbEmpty which is 0, vbLong which is 3, and vbByte which is 17) but then has a 'composite' Enum member (which has a value that is a power of 2), being vbArray which is 8192. VBE_Extras will identify this (and other similar Enums that contain any value that is not a power of 2) as being 'not composite' and will add a Function with the structure of the first of the above two examples.
- For Enums defined in VBA code, when working out whether the Enum is 'composite' or not, the value assigned to the Enum member will be evaluated if it is an expression (rather than just a plain integer number)

Using the Enum members name or value

When adding a Function, whether the Enum members names or values are used (in the Case or If statement, as appropriate for whither the Enum is 'composite' or not) depends on whether the source of the Enum is:

- A Type Library:
    - If the VBA Project has a Project Reference to the relevant Type Library and that Type Library is one of VBA, stdole, MS Office, MS Forms or the library for the host application (e.g. the Excel Type Library if you are using the VBE in Excel), the Enum member names will always be used
    - If the VBA Project has a Project Reference to the relevant Type Library that is any Type Library other than those noted in the above bullet, you will be given the choice of using the Enum member names or the Enum member values (the latter being better if you may subsequently revert to using late binding for the relevant Type Library)
    - If the VBA Project does not have a Project Reference to the Type Library (e.g. if you are using Special binding), the Enum member values will always be used
- Your VBA Project:
    - The Enum member names are always used

When the Enum member names are used, they will be prefixed with the Enum name † … for example:

```
Case VbDayOfWeek.vbSunday: res = "vbSunday" ' 1
```

… instead of …

```
Case vbSunday: res = "vbSunday" ' 1
```

Other info

The name of the Function that is added to the code is determined by the name of the Enum ... it is always GetXxxAsString() where Xxx is the name of the Enum. If an existing Function with that name already exists then it will be updated - so if the Enum is one that is defined in your VBA code and if you update it (e.g. adding, removing, re-ordering or otherwise changing Enum members) then you can just do 'Add a "GetEnumAsString()" Function' again to update the existing Function to match the updated Enum.

The name of the Function's parameter (by default, it is 'value') is defined by the 'Default new parameter name' setting (in the 'Code' tab of the Settings).

The scope of the Function (whether it is Private, Public or default … an Enum cannot have Friend scope) will match that of the Enum itself if the Enum is defined in VBA code, or will have default scope if the Enum is defined in a Type Library.

The number of blank lines added before and after the Function is determined by the 'Blank lines between members' setting (in the 'Code' tab of the Settings).

The code of the Function added by 'Add a "GetEnumAsString()" Function' assumes that you will not pass "invalid" values into the Function. If you do, you may get equally "invalid" results!

† some Enums in some Type Libraries use an 'Alias' for their name ... Enum members cannot be prefixed with such 'aliased' Enum names (they are not recognised by the VBE and will result in compile-time errors) and so the name of the Library itself is used in the GetEnumAsString() Function. Examples of 'aliased' Enums (these from Microsoft Scripting Runtime) are DriveTypeConst, FileAttribute, SpecialFolderConst and StandardStreamTypes (but other Enums in the same Type Library are not 'aliased': CompareMethod, IOMode and Tristate). For anyone reading this that is interested (!), you can see the actual name 'in plain view':

- In your VBA Project, add a Project Reference to the Microsoft Scripting Runtime, then
- Switch on 'show hidden members', and
- Within a procedure or property, type Scripting (which is the 'program name' for the Microsoft Scripting Runtime) then '.' and then, in the VBE's Intellisense popup, scroll down to see the names of the members between 'IOMode' and 'Normal' ...
- The names such as "__MIDL___MIDL_itf_scrrun_0000_0000_0001" are the *actual Enum names* … and these names are legal code when surrounded with square brackets which the VBE will add for you …
- To show that these are the actual names (and are legal code), if you select one in the Intellisense popup to add it to your VBA code and then, after it, type another '.' you will see the Enum member names listed

Consts to Enum / Enum to Consts

If the cursor / selection is one or multiple Consts, converts those Consts to an Enum (one Enum member for each Const).

If the cursor is within an Enum (or the entire selection is an Enum), converts that Enum to Consts (one Const for each Enum member).

This command was primarily added to help when adding Windows API declarations … many constant values come in large groups of individual Consts when a single Enum would be more helpful. This command fulfils that need.

Note that the idea of this command is that the Consts will be converted to an Enum (or vice-versa) immediately after adding them (i.e. before you add other code that references them). Hence, this command does not update references of the Consts (or the Enum) when converting them. If you have already added other code that references them then you may need to manually update that code following the conversion.

Show / hide hidden members

Exactly the same as clicking the same menu option in the VBE's Object Browser window … just that you don't know have to show the Object Browser. You can also:

- Show / hide hidden members, and
- Show hidden members automatically when the VBE starts

… using the relevant items in the 'TypeLibs' tab of the Settings.

Note that this also controls whether Show All Members shows hidden members or not.

Copy as HTML

Copy the selected code as HTML, meaning you can then paste it into (for example) a web page, a blog post, an email or a Word document 'in colour' and using a specific font. Additionally, if pasting into an environment that supports JavaScript (e.g. a web page or blog post) then you can include a 'Copy code' button meaning that readers of the web page / blog post can copy the VBA code with a single click of that button rather than having to manually select and copy the code.

Here's an example without using 'Copy as HTML' (I applied the font name and size after pasting this code):

```
Sub Testing()
    ' this is a comment
    Dim i As Long
    For i = 1 To 10
        Debug.Print Now, "VBE_Extras is awesome"
```

```
    Next i
End Sub
```

And here's an example using 'Copy as HTML' using the VBE's default font name, size and colours:

```
Sub Testing()
    ' this is a comment
    Dim i As Long
    For i = 1 To 10
        Debug.Print Now, "VBE_Extras is awesome"
    Next i
End Sub
```

And another example, this time using some alternative settings defined in the 'Copy as HTML' dialog:

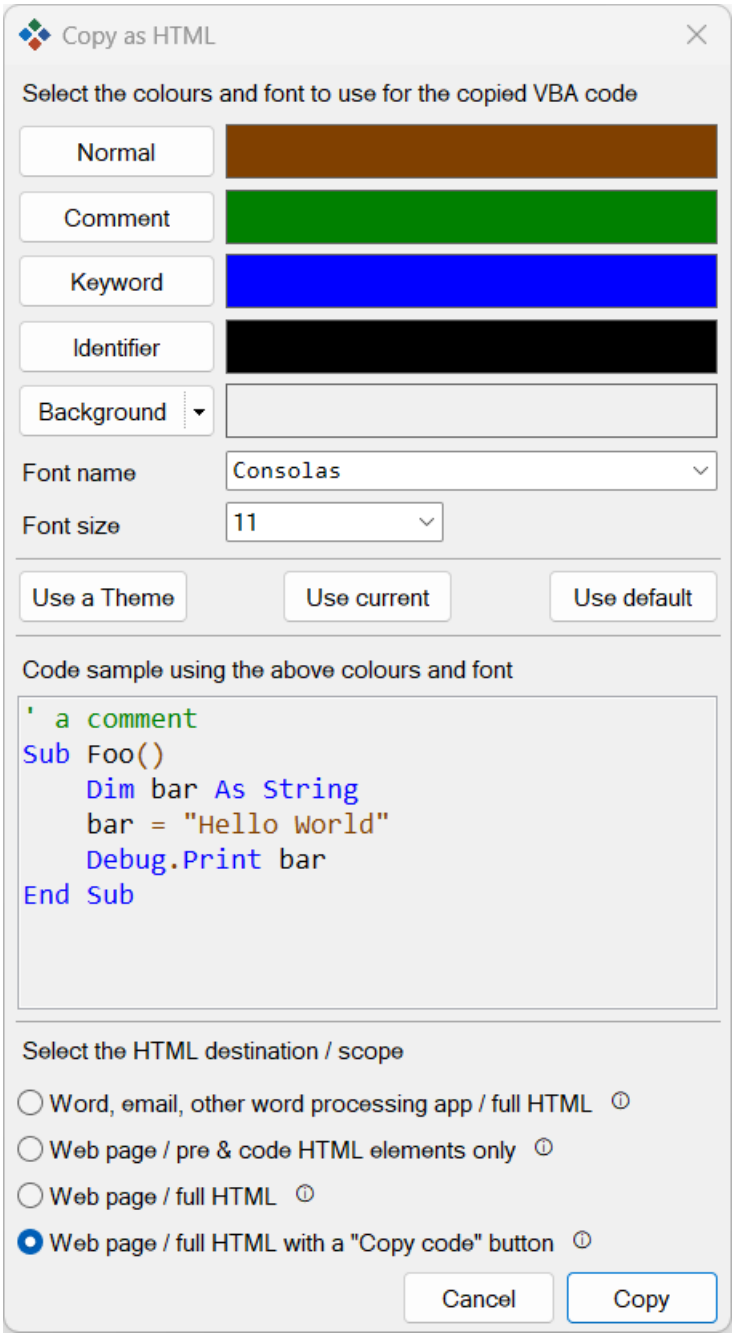
```
Sub Testing()
    ' this is a comment
    Dim i As Long
    For i = 1 To 10
        Debug.Print Now, "VBE_Extras is awesome"
    Next i
End Sub
```

Of course, these examples don't have a 'Copy code' button as they are not in a web page or blog post … to see an example of the 'Copy code' button, see this post: https://www.thevbahelp.com/post/copy-vba-code-in-colour

Note that for the colours of the text to work correctly, the code must be valid / compilable VBA code.

When pasting code 'copied as HTML' into a Word document or into an email, ensure to use the 'Paste as HTML' or 'Keep source formatting' (or similar) paste option … while some applications will automatically recognise that the clipboard is holding HTML and will automatically paste it appropriately … other applications will not do so.

The 'Copy as HTML' dialog



The top half-or-so of the dialog allows you to select the colours ('normal', 'comment', 'keyword', 'identifier' and 'background') to use for the code … 'background' can also be set to transparent. You can then select the font name and size. Alternatives to selecting these individually are to 'Use a Theme' (see Themes), to 'Use current' (i.e. the colours and font currently being used in the VBE) or to 'Use default' (i.e. the colours and font that are used by default in the VBE).

Below that is a code sample showing you an example of what the copied code will look like using the colours and font you have selected.

And below that are the options for the 'HTML destination / scope'. Which one of these you select depends on the destination (e.g. into which application you plan to paste the copied code) and the scope (either 'full HTML' including 'html', 'head' and 'body' elements and style information defined in 'style' elements, or 'pre & code elements only' with styling defined directly in the 'pre' element) of the HTML:

- If you are pasting into anything that is not a web page (or blog post) such as a Word document or an email then you should select the 1st option. This 'scope' optionally allows the use of a CSS file to further define the style of the VBA code in the generated HTML (see Additional files for code style, button style and button texts, below, for details).

- If you are pasting into a web page (or blog post) and cannot include 'html', 'head' and 'body' elements in the generated HTML then select the 2nd option. This scope does not allow for the user of a CSS file to further define the style of the VBA code in the generated HTML.
- If you are pasting into a web page (or blog post) and can include 'html', 'head' and 'body' elements in the generated HTML but do not want a 'Copy code' button then select the 3rd option. The HTML produced by this option may need to be embedded within an iframe element. This 'scope' optionally allows the use of a CSS file to further define the style of the VBA code in the generated HTML (see Additional files for code style, button style and button texts, below, for details).
- If you are pasting into a web page (or blog post) and can include 'html', 'head' and 'body' elements in the generated HTML and want to include a 'Copy code' button * then select the 4th option. As with the 3rd option, this 'scope' may need to be embedded within an iframe element. This 'scope' optionally allows the use of CSS files to further define the style of the VBA code in the generated HTML , the style of the 'Copy code' button and the text used in the 'Copy code' button (see Additional files for code style, button style and button texts, below, for details). Note that the 'Copy code' button uses JavaScript and so whatever application is being used to display the code must support JavaScript. Of course, all modern web browsers support JavaScript – though it should be noted that all can also be configured by end-users to disable JavaScript.

* The VBA code copied to the clipboard by the 'Copy code' button is always plain text with the intention that end-users will then paste this code directly into a Module of a Project in the VBE ... as such, it would not be helpful for it to contain HTML elements!

Additional files for code style, button style and button texts

The 'Copy as HTML' dialog allows customisation of the colours and font used for the VBA code … but nothing else. CSS (which is used to style HTML) allows for a vast array of styling properties (see https://www.w3schools.com/css/) and, over time, these properties are updated. As I do want to allow for other CSS properties to be used (but don't want to spend crazy amounts of time creating a dialog that allows every individual CSS property / value combination to be selected … and then maintain that dialog in line with CSS standards) then the approach I have taken to allow further customisation of the copied VBA code and of the 'Copy code' button itself is to use specific files containing CSS properties and values. Using these files, you can apply almost any styling you want.

Additionally, you can define alternative text to use for the 'Copy code' button … both in its default 'Copy code' state and in its 'Copied' state (which is briefly shown when it is clicked).

For all of these options, a specific text (i.e. .txt) file most be created and saved in the "C:\Users\%username%\AppData\Local\VBE_Extras\settings\" folder on your device (the device that VBE_Extras is installed on) where "%username%" is your username. This folder will already exist on your device (you do not have to create it). The file must be present on your device at the time you click the 'Copy' button in the 'Copy as HTML' dialog.

To customise the styling of the VBA code

Create a file named "CopyAsHtmlCodeStyle.txt" and save it in the above noted folder. The file should contain valid CSS properties and values only. To learn more about the properties and values for CSS, see the above link. For example, a file with these CSS properties and values …

```
border-style: solid;
border-width: 1px;
border-color: red;
padding: 10px;
margin-left: 30px;
background-color: azure;
```

Results in the addition of a solid red border, some padding and a margin on the left …

```
Sub Testing()
    ' this is a comment
    Dim i As Long
    For i = 1 To 10
        Debug.Print Now, "VBE_Extras is awesome"
    Next i
End Sub
```

Note that when using such a file to customise the styling of the VBA code:

- The text / foreground colours selected in the 'Copy as HTML' dialog (i.e. 'normal', 'comment', 'keyword' and 'identifier') are used, and
- The font name and size selected in the 'Copy as HTML' dialog are used, but
- The 'background' colour is not … if you do not specify a particular `background-color` CSS property (in the above example CSS it is set to the value `azure`) then the background colour will be inherited from the styling of the parent HTML element

To customise the styling of the 'Copy code' button

Create a file named "CopyAsHtmlButtonStyle.txt" and save it in the above noted folder. The file should contain valid CSS properties and values only. To learn more about the properties and values for CSS, see the above link. For example, a file with these CSS properties and values …

```
background-color: #D0D0D0;
border: none;
color: #217346;
padding: 4px;
text-align: center;
text-decoration: none;
display: inline-block;
font-size: 11px;
font-weight: bold;
margin: 2px;
cursor: pointer;
```

… replicates the default button (grey background, green text etc) which, of course, you can customise … whereas a file with these CSS properties and values …

```
background-color: #FFFF80;
border: 2px dotted red;
color: #000000;
padding: 2px;
text-align: center;
text-decoration: none;
display: block;
font-size: 14px;
font-weight: italic;
margin-top: 2px;
margin-bottom: 2px;
margin-left: auto;
margin-right: 0;
cursor: pointer;
```

… makes the button have a yellow background with a red dotted border, a larger font and is aligned to the right (whereas the default button is aligned to the left). To see examples of the 'Copy code' button, see the blog post https://www.thevbahelp.com/post/copy-vba-code-in-colour.

To customise the text of the 'Copy code' button

By default, the 'Copy code' button uses the text 'Copy code' in its default (waiting to be clicked) state and 'Copied' (briefly) once clicked (before reverting back to the default text). You can change this text (in either or both states) using files named "CopyAsHtmlButtonTextNormal.txt" (for the default state) and "CopyAsHtmlButtonTextCopied.txt" (for the copied state). The files should contain the to-be-used text (no speech marks etc) and can include any Unicode (UTF8) characters. The files should be saved in the above noted folder.

Again, for examples, see the blog post https://www.thevbahelp.com/post/copy-vba-code-in-colour.

If you are having problems with 'Copy as HTML'

Problem	Comment / solution
The colouring of code (i.e. into 'normal', 'comment', 'keyword' or 'identifier') does not work correctly.	The copied VBA code must compile successfully for the colouring to be applied correctly.
When pasting the copied VBA code into Word or an email, the HTML, CSS and / or JavaScript itself is visible.	Use the 'Paste as HTML' or 'Keep source formatting' (or similar) paste option … while some applications will automatically recognise that the clipboard is holding HTML and will automatically paste it appropriately … other applications will not do so.

The 'Copy code' button just appears as plain text when pasting the copied VBA code into Word or an email.	The 'Copy code' button only works in web pages (or blog posts).
End users report that the 'Copy code' button is missing or does not work.	Likely JavaScript is disabled in the user's browser … JavaScript must be enabled for the 'Copy code' button to work.
The VBA code is not picking up the custom CSS properties and values defined in a file / the 'Copy code' button is not picking up the custom CSS properties and values defined in a file.	You must be using one of the 'full HTML' options to copy the code (not the 'pre & code elements only' option). The CSS file must be named exactly as noted above and saved in exactly the right folder. The file most contain valid CSS properties and values only (with a colon between the name of the property and the value, and a semi-colon after the value). If you have many CSS properties and values in one file, try using only one CSS property and value and then adding others one-at-a-time to see which one may be causing a problem.
The 'Copy code' button is not picking up the custom text defined in a file.	The text file(s) must be named exactly as noted above and saved in exactly the right folder. Do not include 'special characters': quotation mark (speech mark); apostrophe; ampersand; less-than; greater-than.

Add Windows API declarations

Note that you should always make a backup of your VBA project (see Backup and Restore) before running code that uses Windows API declaration(s) … whether those declarations were added using VBE_Extras or not. Using Windows API declarations incorrectly can result in an immediate crash of the host application and data loss.

I have not personally tested each of the declarations that VBE_Extras can add to your VBA code (I have tested many of them … but not all of them!) and so I cannot provide a guarantee that every declaration is correct. A large number of the declarations that VBE_Extras can add come from Microsoft's "Win32API_PtrSafe.TXT" file … this file includes many errors (which are fixed within VBE_Extras … for those errors that I am aware of) and includes a number of deprecated declarations (which are excluded from those that VBE_Extras will add … as far as I know them).

Additionally, even if a declaration is correct, passing it invalid data or otherwise using it incorrectly can also result in an immediate crash of the host application.

VBE_Extras will help you to identify and add (to your VBA code) Windows API declarations … VBE_Extras is aware of the VBA declarations for:

- 2,079 Functions (counting Functions that include both an ANSI and Wide variant as 1 … if counted as 2 discrete Functions then that would add several hundred more)
- 518 Types
- 52 Enums
- 11,161 Consts

In the 'Windows API declarations' dialog, you can:

- View a list of all Windows API declarations that VBE_Extras is aware of – the list includes:
  - The type and name of the declaration
  - It's 'bitness' (some declarations are 32 or 64 bit only; the majority are 'both')
  - If the declaration is ANSI-specific, whether a Wide (i.e. Unicode) variant is also available
  - The number of implementation examples available, if any (see Example usages of Windows API declarations)
  - The full code of the declaration - the text in the 'Code' column is fixed (for example, the 'Code' column always shows the ANSI version of the declaration), however, the tooltip for this column is updated dynamically based on other selections made in this dialog and also in the Settings
- Search / filter for Windows API declarations – any search text that you enter:
  - Always searches by declaration name
  - Always searches the description of any implementation example for the Windows API declaration (not all declarations have examples)

    - For Functions, Types and Enums (not Consts), the (English-language) text description for the declaration is searched … for example, a search for "class*name" (see Appendix 8 – wildcards in 'filter text boxes') will display 3 Functions: FindWindowA, FindWindowExA and GetClassNameA (the words "class" and "name" are included in the description of both FindWindowA and FindWindowExA; those two words are included in the name of GetClassNameA)
- Filter by Windows API declaration type – that is, show all declarations or only show Functions, Types, Enums or Consts

Any combination of Windows API declaration(s) can then be selected. If:
- You select a 32 bit declaration, you will be asked whether you want the 64 bit equivalent to also be selected (and vice-versa)
- You select a Function that is dependent on a Type or an Enum (e.g. a Function that take a Type as a parameter) then you will be asked if you want the Type to also be selected
    - The same also happens for Types that are dependent on other Types; Types that are dependent on Consts; and Consts that are dependent on other Consts (this is recursive so that, for example, if you select a Function that depends on a Type that is itself dependent on another Type that is dependent on a Const, you will be asked if you want all of those dependencies to be selected)
- You select a Const that is part of a related group of Consts e.g. if you select the ABE_RIGHT Const, you will be asked if you want the 3 other related Consts (ABE_BOTTOM, ABE_LEFT and ABE_TOP) to also be selected … the grouping of related Consts is not purely text based (for instance, there are 3 discrete groups of Consts that all start with "SW_")
    - Having added a related group of Consts, you can optionally use the Consts to Enum / Enum to Consts command to convert a group of related Consts into a single Enum (with one Enum member for each Const)

You can then choose various options for the format of the Windows API declarations. Options exist for:
- Adding only a VBA7 declaration or both a VBA7 and a pre-VBA7 (i.e. VBA6) declaration using appropriate conditional compilation
- Adding the ANSI or Wide (i.e. Unicode) variant of any Functions that have both variants
- Whether to convert the value type of any parameters that are Types to Long (for pre-VBA7) or LongPtr (for VBA7) …
    - The reason for doing this is to allow pointers to the Types (using VarPtr) to be passed to the Windows API Function/Sub instead of the Type itself (which then avoids VBA's automatic conversion of Strings from Unicode into ANSI when passing them to the Windows API)
    - If you select 'Based on Type members' then parameters with value types that are Types will only be converted to Long/Ptr if the Type has a member that is a String or if the Type is dependent on another Type that itself has a member that is a String; otherwise the value type will be left as the Type
    - Note that when a value type is converted from a Type to a Long/Ptr then the parameter will also be changed from ByRef (which is required for Types) to ByVal so the actual pointer is passed, not a reference to the pointer
- Whether to convert the value type of any String parameters to Long (for pre-VBA7) or LongPtr (for VBA7) …
    - The reason for doing this is to allow a pointer to the String (using StrPtr) to be passed to the Windows API Function/Sub instead of the String itself (which then avoids VBA's automatic conversion of Strings from Unicode into ANSI when passing them to the Windows API)
    - For examples of using StrPtr to pass a pointer instead of the actual String, see the blog posts 'MsgBox with a timeout' at https://www.thevbahelp.com/post/msgbox-with-a-timeout or 'MsgBox with custom button texts' at https://www.thevbahelp.com/post/msgbox-with-custom-buttons
- Adding the declaration(s) as Private, Public or 'default'
- If and how to split the line for lengthy declarations of Functions … selecting 'Line length' means that the 'line split' options as set in the 'Indent/Split' tab of the Settings will be applied

Additionally, the 'line indent' options (as set in the 'Indent/Split' tab of the Settings) will be used for any line indentation when adding the Windows API declaration(s) to the active Module.

You can then add the selected Windows API declaration(s) to the active Module:
- As a discrete block of code with conditional compilation directives for VBA7 and / or Win64 added where required (this option is always available), or
- Merged with any existing Windows API declarations already in the Module … that is, new declarations will be added within the appropriate existing conditional compilation directives for VBA7 and / or Win64 (this option is only available when VBE_Extras detects existing Windows API declarations in the active Module and when the directives can be correctly parsed by VBE_Extras)
- Alternatively, if you would rather add the Windows API declaration(s) yourself (or, otherwise you just want the text of the declaration(s)), you can use the 'Text' button.

VBE_Extras also stores details of which Windows API declaration(s) you have selected and allows you to quickly add them again using the 'Recent' button.

There is a right-click context menu for the list of Windows API declarations … depending on which declaration is clicked, there are options to:

- Open the Microsoft web page for the declaration (the web page will always be the English language web page)
- Copy the URL of the Microsoft web page for the declaration (the URL will always be the English language URL)
- Copy the description (from the Microsoft web page) of the declaration
- Copy the declaration (either the VBA7 version, the pre-VBA7 version, or both)
- View Example usages of Windows API declarations, if any, for the declaration

CHAR / WCHAR

Some Windows API Functions/Subs (in parameters) and Types (in members) use CHAR in the ANSI ("A") variant of the declaration and WCHAR in the Unicode / wide ("W") variant of the declaration. Note that a CHAR uses 1 byte of memory whereas a WCHAR uses 2 bytes of memory.

As such, when VBE_Extras adds the ANSI ("A") variant of a Function/Sub (or a Type that is used from the ANSI ("A") variant of a Function/Sub), then:

- A parameter / member that is declared as a singular CHAR will be added to your VBA code as a Byte
- A parameter / member that is declared as an array of CHARs will be added to your VBA code as an array of Bytes

And when VBE_Extras adds the Unicode / wide ("W") variant of a Function/Sub (or a Type that is used from the Unicode / wide ("W") variant of a Function/Sub), then:

- A parameter / member that is declared as a singular WCHAR will be added to your VBA code as an Integer
- A parameter / member that is declared as an array of WCHARs, you will be given the option of adding it to your VBA code as either:
    - An array of Bytes (with the bounds increased to double the number of bytes)
    - An array of Integers (with the same bounds)
    - A fixed-length String (with a length matching the bounds of the array)

Example usages of Windows API declarations

VBE_Extras is aware of examples for 503 Functions plus examples for a number of Types, Enums and Consts (i.e. so not all declarations that VBE_Extras is aware of have examples ... but the bulk of examples tend to be for the most commonly used declarations).

Examples can be accessed in two ways, both via the 'Windows API declarations' dialog:

- Right-clicking on a specific Windows API declaration will allow access to any example(s) that include that declaration
- Clicking on the 'Examples' button shows a dialog listing all examples … you can then filter the list to identify an example

119 of the examples are online and 132 examples are stored locally on your device (within VBE_Extras). The online examples are on VBA-related sites ... either Q&A web sites (such as StackOverflow, MrExcel) or blog posts. Other than those examples on my own website (https://www.thevbahelp.com/blog), I have no direct control over the content (which may change over time or even be removed) so please use caution if implementing any code from the site.

Example usages are intended to assist with implementation of the Windows API declaration in your code. The example(s) for a particular declaration may or may not relate to how you want to use the declaration … but should give some hint as to its usage which is often not obvious from the 'bare bones' declaration and it's Microsoft web page alone. Please don't consider the example usages to be definitive (certainly for the more regularly used Windows API declarations there will be other example available online) … they are intended to allow you to quickly find an example to assist with implementation in your own code.

Also please note that the code of the Windows API declaration(s) in an example may not match the code of the Windows API declaration(s) shown in the 'Windows API declarations' dialog (and added to your VBA code by VBE_Extras). This doesn't indicate a problem with the example or with the declarations in VBE_Extras. For example:

- For a Function, if an "Alias" is specified then the declared name of the Function does not have to match the name on the Microsoft web page
- For Function parameters:
    - The names of the parameters do not have to match the names on the Microsoft web page

    - The value types of the parameters can be different … for example, when using the "W" (ie Unicode) version of Functions, it is common to use Long/Ptr instead of String … and often Any can be used to allow passing different value types
    - The correct use of ByVal or ByRef (the default in VBA, if omitted) is crucial … but sometimes either can be used depending on what is being passed … for example if passing a Type then it must be passed ByRef, but if passing a pointer to a Type (ie using VarPtr) then ByVal must be used
- The name of a Type, Enum or Const (and the names of the members of a Type or Enum) do not have to match the names on the Microsoft web page … however, for Types the order and value types of its members must be exactly correct; and for Enums and Consts, the value(s) must be exactly correct

Note about the Windows API declarations that VBE_Extras 'knows about'

The Windows API and the declarations it contains evolves constantly. New Functions, Types, Enums and Consts are regularly added (for instance, it is common for new Consts to be added so that existing Functions have a new behaviour). And occasionally Functions, Types, Enums and Consts are deprecated.

While I will update the VBE_Extras data relating to Windows API declaration when I am aware of something being incorrect or changing, I do not intend to make it my life's work to maintain the data in line with Windows itself. To ensure you are always using a Function correctly, **read the Microsoft documentation** … the web page for the Microsoft documentation for every Windows API Function that VBE_Extras 'knows about' (along with a good number of the Types, Enums and Consts) can be opened from the 'Windows API declarations' dialog or can be opened directly from the VBE using F1 to view Windows API web pages.

Note about possible issues with the order that Windows API declarations are added to Modules

It is possible that, when adding Type and/or Const declarations (especially when merging into existing Windows API declarations with other Types and/or Consts), they may not be added in the optimum order resulting in a 'forward dependency' compile-time error ... the issue here is with the order of the declarations within the Module, the issue is not with the validity of the declaration itself. Typically this can happen when:

- A Type is dependent on another Type (as the 'dependee' Type must appear in the Module before the 'dependent' Type ... this would result in a compile-time error "Forward reference to user-defined type"), or
- A Const is dependent on another Const (as the 'dependee' Const must appear in the Module before the 'dependent' Const ... this would result in a compile-time error "Constant expression required")

In both cases, the solution is simple:

1. Having compiled your code, the VBE will move the cursor so the 'dependent' declaration that is causing the error (and will show one of the above two error texts depending on whether the problem is with a Type or with a Const)
2. You can then identify the 'dependee' Type or Const as it's name will be included in the declaration of the 'dependent' Type or Const
3. You can then move the declaration of that 'dependee' Type or Const to be before (i.e. above) the declaration of the 'dependent' Type or Const (i.e. to be before the declaration that is causing the error)

Project stats

Provides details for each Module in the active Project, including number of declaration lines, number of code lines, number of comment lines, number of Procedures and Properties

View

Previous Lists for this Project

Allows previously-shown dialogs with lists of declarations, references, literal values, tasks and calls to be immediately re-shown. This command is 'single-Project' in that the Previous Lists are available per-Project … if you activate a different Project then the Previous Lists shown when that Project was active will be shown.

All information shown is 'as it was' when the original list was shown e.g. the numbers of declarations, references, literal values, tasks and calls and their locations (line / column numbers) are not updated if code has been edited since the list was originally shown … the idea is that this is a quick way of re-showing a specific list and so is useful when generating a list can take some

time. As such, some functionality is limited, for example selecting 'Go to' will go to the right location if it is available or will go to the nearest possible location otherwise (e.g. if lines have been deleted from or added to a Module).

Previous lists are stored per-Project. If you navigate to a different Project, the previous lists available to be shown will be those for that Project. Lists are not deleted when you navigate away from a Project, so you can navigate back to a Project and to see the previous lists for that Project. However, previous lists are deleted when a Project is closed or if it is saved with a different filename or in a different folder.

There is a maximum number of each type of list that is stored. When this maximum is exceeded then the oldest lists will be automatically deleted.

Specific previous lists will be automatically deleted if they become 'invalid' for example if a Module is deleted from your Project and that list included an item that was in that Module or referred to that Module.

FullScreen

Toggle 'FullScreen' on or off – this hides (when going into FullScreen) / shows (when leaving FullScreen) the 'dockable' windows (Project, Properties, Immediate, Locals, Watches and Object Browser) to leave the (one, if you have multiple displayed e.g. 'tiled' or 'cascaded') active Code window or UserForm designer window in FullScreen (menus and toolbars are still visible). You can choose to not hide some (but not all) of the dockable windows in the Settings.

When you select FullScreen: if any of the dockable windows (other than those you may have chosen to 'not hide') are not showing then the VBE is considered to be in FullScreen mode and all dockable windows will be shown; if all dockable windows are showing then the VBE will go into FullScreen mode.

FullScreen mode being on or off is per-application (if you have multiple instances of the VBE open). Selecting which of the dockable windows are hidden in FullScreen mode is across all VBE-hosting applications but will only take effect in another application's VBE when that applications is re-started.

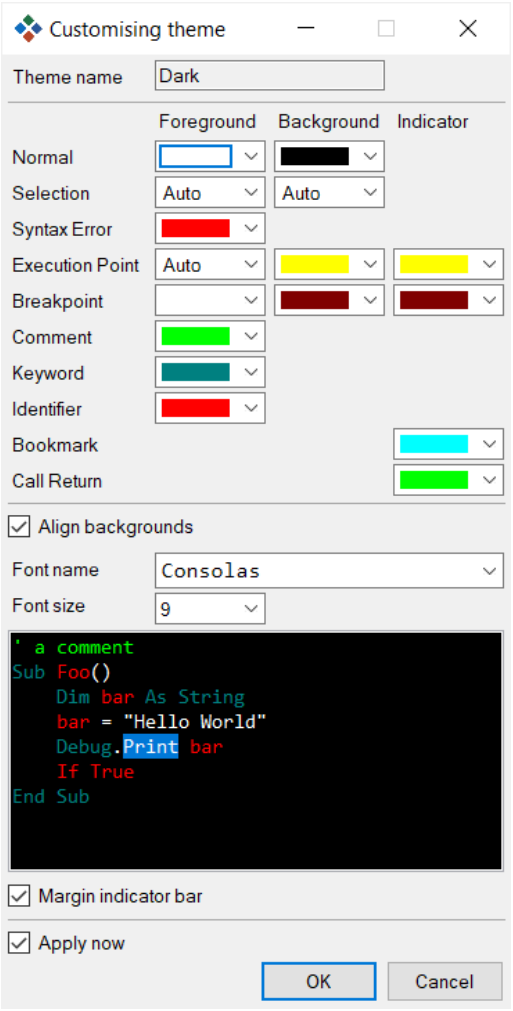
Note that whether a 'dockable' window is currently set to dock or not is set in the VBE Options (Tools > Options > Docking).

Themes

The Themes dialog is an improved version of the 'Editor Format' tab of the VBE's built-in 'Options' dialog. The Themes dialog allows you to choose the same colours (text, background and 'indicators'), font name, font size and presence of the 'margin indicator bar' (MIB), but in a much more logical and user-friendly manner.

For example, the Themes dialog:

- Only allows you to select a colour when that resulting colour can actually be shown (for example, the built-in Options dialog allows you to select a foreground, i.e. text, colour for a Bookmark when a bookmark is only ever indicated by an indicator in the MIB; the Options dialog allows you to select an indicator colour for Keyword text when Keyword text does not have an indicator)
- Allows you to select one colour as the background colour for all of the 'main' text types … if 'Align backgrounds' is selected then the background colour chosen for 'Normal' text will also act as the background colour for text that is for Syntax errors, comments, keywords and identifiers
- Includes a sample code image (see image to the right) with a greater variety of text-types allowing you to see immediately the result of the colour and font changes you have made (the word 'Print' in the image is the colours used when text is selected)
- Has a font name drop-down that shows fonts *in that font* and indicates which fonts work with 'new-style' Highlighting
- Allows you to save your colour, font and MIB visibility selections as 'Themes' … and you can create as many Themes as you want … meaning you can switch between entire Themes with a few keystrokes / mouse clicks



As with the Options dialog:

- The colours and font apply only to the main code window and to the Immediate window, if visible … the MIB visibility applies only to the main code window
- The colours, font and MIB visibility that you select for the VBE in any one host application will also apply to VBE's in any other host application … but, if that/those other VBE(s) are already opened, it will only apply once you close and re-open the other VBE(s) and host application(s) … see below for details as to why

Technical stuff – read only if you're interested or if the Themes dialog isn't doing what you might think it should!

- Themes are applied to the VBE by automating the Options dialog (you will see this in action when you apply a Theme) ... if you move focus to another window while a Theme is being applied, you will find the Theme is not applied fully
- When you add a new Theme based on the 'current settings', the current settings are also read by automating the Options dialog ... again, if you move focus to another window while the current settings are being read, you will find the Theme shown in the Theme dialog is not entirely correct
- Ultimately, the colour, font and MIB visibility are stored in the Registry. You may wonder why VBE_Extras bothers with automating the Options dialog instead of just updating the values held in the relevant Registry … the main reasons are:
    - The VBE only reads colour, font and MIB visibility from the Registry when the VBE is first loading so changing the values in the Registry would make no difference while the VBE is running … you would have to close and re-open the host application for the changes to take effect
    - In VBE 7.1 (32 and 64 bit … and perhaps in earlier versions), the VBE fails to save colours for the indicators … it simply doesn't create the relevant Registry key so any custom colours you choose for indicators will be honoured only while the VBE / host application is running (you can test this by choosing custom colours, closing and then re-opening the VBE and the host application: you will find the indicator colours have returned to their defaults). VBE_Extras resolves this for you, when you apply changes as a Theme, by creating the relevant Registry key
- For the reason above (the VBE only reads colour, font and MIB visibility from the Registry keys when the VBE is first loading), if you change the Theme in the VBE for one host app, it won't change the Theme in any other open VBEs in other host applications unless you close and re-open both that other VBE and its host application

Interaction with VBE_Colours

VBE_Colours is a separate application (available on the https://www.thevbahelp.com/vbe-colours website) that allows you to select custom colours to be displayed in the VBE. Unlike the Themes command of VBE_Extras (which allows you to select from the built-in VBE colours), VBE_Colours actually allows you to select different colours.

Those custom colours always be honoured in the main code window and in the Immediate window when applied as a Theme by VBE_Extras so long as you are using VBE7.1 (that is Office 2013+), 32 or 64 bit.

If you have installed VBE_Colours, Theme "(VBE_Colours custom)" will be shown automatically in the relevant Themes dialogs … applying this Theme acts the same as setting the registry values to 'custom' from within VBE_Colours.

If you are thinking 'custom colours'??!! Yes, it is possible. VBE_Colours will allow you to change the actual colours … that is, not select from the pre-existing 16 colours that the VBE makes available, you can actually select different colours. See the information at the VBE_Colours link, above, for more info.

Help

Check for update

Manually check whether there is an updated version of VBE_Extras. VBE_Extras will also do this periodically in the background.

Help

Access the latest version of this VBE_Extras guide.

View blog post

View a VBE_Extras-related blog post … i.e. via https://www.thevbahelp.com/blog/categories/vbe_extras

What's new

Show what is new in VBE_Extras … the same as Appendix 2 – Update history

Report a bug

If you find a bug, please report it! But remember that VBE_Extras was developed by a single developer in his spare time … so please be kind!

Reporting via GitHub is preferred (it allows you, before raising a bug report, to check if the same bug has already been raised; additionally, you will automatically get updates on the progress of your bug) but via email is fine!

Licence

Includes options to:
- Buy a licence for VBE_Extras (via the website)
- Activate a licence (i.e. once you have bought a licence)
- Remove a licence that you have previously activated

Note that VBE_Extras comes with a free trial period – you do not have to buy a licence immediately. Additionally, some VBE_Extras commands are "free" (you can continue to use them beyond the end of the trial period without purchasing a licence … see https://www.thevbahelp.com/vbe-extras for details).

Settings (Backup/save, Restore/load, To defaults)

Backup settings, restore settings, and reset settings to default. The following categories of settings are included:
- Settings made in the Settings dialog
- Themes
- The majority of the options selected on the various other dialogs that can be shown by VBE_Extras (e.g. the dialogs that show lists of declarations, lists of references, the rename dialog etc)

Conditional Compilation Arguments

For more on conditional compilation generally, see Conditional compilation

The Project needs to have been saved at least once for the 'Conditional Compilation Arguments' commands to work. Note that the 'Conditional Compilation Arguments' commands are multi-Project in that CCAs for multiple Projects can be stored simultaneously by VBE_Extras and they will be used only when parsing the relevant Project.

Once CCAs have been read-in or entered manually (see the next two sections), VBE_Extras can automatically store and restore them when your Project is closed and re-opened … for details, see Store / restore Conditional Compilation Arguments.

Update CCAs for this Project automatically

VBE_Extras will read in the Conditional Compilation Arguments from the Project Properties window for the active Project and apply those to future actions (such as finding declarations and references). If you change the CCAs and want VBE_Extras to be 'aware' of the new CCAs, you must call this option (or Update CCAs for this Project manually) again.

Update CCAs for this Project manually

You can enter the CCAs for the active Project manually e.g. if you want to use different values than those in the Project Properties window or if the option to Update CCAs for this Project automatically fails for any reason.

Clear CCAs for this Project

Clear the CCAs for this Project.

Backup and Restore

Backup this Project

Create a backup copy of the active VBA Project in your chosen backup folder … the location of the backup folder and whether a 'description' is included can be changed in the 'App' tab of the Settings.

The VBA Project needs to have been saved at least once for the 'Backup this Project' command to work.

Create restore point

Manually create a restore point for the active VBA Project … see Restore for further details.

Restore

Restore a previously-created restore point for the active VBA Project. The code (see 'Scope', below, for details of what is included in a restore point) of the VBA Project will be reverted to that at the time of the restore point … this includes adding any Modules that you deleted since the restore point was created, and deleting any Modules that you added since the restore point was created (but see Document Modules in host applications other than Access) … restore points are like a limited form of Version Control.

Restore points can be created:

- Manually via the Create restore point menu item, and
- Automatically in advance of refactoring commands … if-and-when automatic restore points are created is controlled by the settings in the 'Restore Points' tab of the Settings

Restore points are only ever for one VBA Project … even if the VBA Project has Project References to another VBA Project: only the active VBA Project is included in the restore point; and only the active VBA Project will be updated when restoring (if your Project has Project References to other Projects and if you are using a command that makes changes across multiple VBA Projects, consider manually creating restore points for the other affected Projects). Additionally, when restoring, you will only be shown restore points for the relevant VBA Project (i.e. if you are working on multiple VBA Projects and creating restore points, when restoring you will only be shown the restore points for the VBA Project that is active in the VBE).

Note that when you select a restore point, you will always first be shown a Diff analysis before any restore takes place giving you the opportunity to review the details of what will be changed and then to either proceed with the restore or cancel it (and select a different restore point if desired).

Restore points are intended to be temporary and will be regularly deleted (to control the duration that they are retained for, see the 'Restore Points' tab of the Settings). For "longer term" restore points, use Version Control.

Changing the name or location of a file (Workbook, Database etc) that contains the VBA Project will mean that any previously-created restore points will be deleted.

Scope

The scope of what is included in a restore point (and so what is restored) is as follows:

- In host applications other than Access:
  - VBA code … in all types of Modules
  - VBA attributes … in Standard, Class and UserForm Modules
  - UI elements in UserForms
- In Access, in addition to the above:
  - UI elements in Forms and Reports
  - VBA attributes in Forms and Reports

The scope differs from Version Control in that Project References and RibbonX 'customUI' file(s) are not included.

Which commands are restore points automatically created for

If creating restore points is switched on for 'complex' refactoring commands and if the 'maximum frequency' has passed, then a restore point will be created before:

- Rename (code at cursor, this Module, this Project, this UserForm, this Control)
- Add Properties †
- Add Factory
- Extract to Constant
- Implement Interface
- Extract Interface
- Make 'Option Explicit'
- Update parameters (add, promote, remove, demote and reorder)
- Add / update / remove line numbers
- Indent lines
- Split lines
- Attributes – adding / updating / removing
- Replace all Modules in this Project
- Add Windows API declarations
- Consts to Enum / Enum to Consts
- Fix case

If creating restore points is switched on for 'minor' refactoring commands and if the 'maximum frequency' has passed, then a restore point will also be created before:

- Add Properties †
- Qualify reference
- Comment / uncomment lines
- Move lines up / down
- Insert '@EntryPoint'
- Add Enum from Type Library
- Add a "GetEnumAsString()" Function

The differentiating factor between 'complex' and 'minor' refactoring commands is that 'complex' refactoring commands can make multiple changes to your code, potentially across more than 1 Module (or even across more than 1 Project in some cases)

whereas 'minor' refactoring commands will only ever make a single change in a single Module and that change can be easily undone (either using the VBE's 'Undo' button or by simply reversing the change you made e.g. by moving lines in the other direction or by uncommenting commented-out lines).

† Add Properties is a 'minor' refactoring command unless: the name of the added Property(s) clashes with the name of existing members in the same Project AND that clash of names requires one or more references to be qualified with the relevant Module name

For clarity, restore points are not automatically created before:

- AutoText (including auto closing pairs) ... the delay would interrupt the 'typing flow'
- Version Control … if you are importing an earlier version of your VBA Project, then you have decided that the existing version of your VBA Project is no longer required (or, if not sure, Backup this Project first)
- Update Project References … as Project References are not included in restore points, there would appear to be no change when restoring
- Clean this Project … you would only run this command if your VBA Project was corrupt (if not sure, Backup this Project first)

Other information

See Appendix 6 – relating to Version Control and Restore for sections on:

- Relevant to host applications that support UserForms
- Document Modules in host applications other than Access
- Access-specific
- Case-only Module changes
- Other info

Settings

Change settings for VBE_Extras. Settings for:

- Task Items - see Tasks
- Rename - see Rename code at cursor
- Projects - see Referenced VBA Projects
- Shortcut Keys - switch on/off and redefine which keyboard shortcuts operate which VBE_Extras commands
- Run - see Sub to run and Immediate window
- Code - see various of the commands in Refactor
- Indent/Split - see Indent lines and Split lines
- FullScreen - see FullScreen
- Highlighting - see Highlight
- Version Control - see Version Control
- Restore Points - see Restore
- AutoText - see AutoText, including Auto closing pairs
- TypeLibs – settings related to Type Libraries including Show All Members and Special binding
- UI - various user-interface related settings
- Other - which includes settings for:
    - Reference order - see References for code at cursor
    - Whether to match conditional compilation directives - see Conditional compilation
    - Whether mouse-click commands are enabled - see Mouse-click commands
    - Orphans - see Orphan declarations
    - Line numbering - see Add / update line numbers and Remove line numbers
    - Option Explicit - see Option Explicit
    - Names - see Go to Name with this name
    - Comment / Uncomment - see Comment / Uncomment lines
    - Move lines up/down - see Move lines up / down
    - Alternative VBE window navigation keys - see Alternative VBE window navigation keys
    - Store / restore cursor location and CCAs - see Store / restore
    - Types of declarations (in this Module / Project) to show in lists – see List declarations in this Module and List declarations in this Project

- App – which includes settings for:
  - Allowing VBE_Extras to automatically check for updates
  - Logs - set the logging level, view the logs
  - Other tools to support debugging
  - Changing the main menu 'x' accelerator key - see Appendix 4 – changing the main menu's 'x' accelerator key
  - Backup folder - see Backup this Project

To easily locate a specific setting by its name, click on 'Find Setting' then either type in the 'filter' box (including "?" and "*" wildcards … see Appendix 8 – wildcards in 'filter text boxes') or scroll the list.

See also Settings (Backup/save, Restore/load, To defaults)

About VBE_Extras

Information about VBE_Extras and the environment it is running in on your device.

Commands not available in the main menu

Some of these commands are also available in other menus (though not in the main menu), in the Toolbar and/or using shortcut keys.

References of this Project

Available in the Project window context menu (when the Project name is clicked).

Same as References for code at cursor but for the Project clicked in the Project window.

References of this Module

Available in the Project window context menu (when an individual Module name is clicked).

Same as References for code at cursor but for the Module (document, standard, Class or UserForm) clicked in the Project window.

View as text file

Available in the Project window context menu (when an individual Module name is clicked).

For a Module, create and display a text file with a single click.

Compare Project

Available in the Project window context menu (when the Project name is clicked).

View a report of the differences from the current version of a Project (i.e. the Project clicked in the VBE) to either:
- A previously exported Project (for example, a Project exported using Version Control), or
- Another Project in the VBE (the other Project must be unlocked and must have been saved at least once)

See Diff analysis for further details.

Compare Module

Available in the Project window context menu (when an individual Module name is clicked).

View a report of the differences from the current version of a Module (i.e. the Module clicked in the VBE) to either:
- A previously exported version of the Module, or
- Another Module of the same type in the same Project, or

- Another Module of the same type in another Project (the other Project must be unlocked)

See Diff analysis for further details.

Diff analysis

For an entire Project, the diff analysis shows:

- Which Modules have been added, deleted or updated, and from that you can drill down to:
  - For added or deleted Modules, view the code of the Module
  - For updated Modules, view a report of the differences at the 'Module member' † level (view added, deleted or updated members) … again you can drill down to:
    - For added or deleted members, view the code of the member
    - For updated members, view the code of the two versions compared … see 'Code compare', below
- If an export was created via Version Control (not a restore point) using VBE_Extras version 1.4.6.0 or greater, which Project References have been added or deleted
- If an export was created via Version Control (not a restore point) using VBE_Extras version 1.4.7.0 or greater and if the 'RibbonX – export / compare-on-import' setting was checked at the time of the export, which RibbonX 'customUI' XML file(s) have been added, deleted or updated, and from that you can drill down to:
  - For added or deleted 'customUI' XML files, view the RibbonX
  - For updated 'customUI' XML files, view the XML code of the two versions compared … see 'Code compare', below

† breakdown of 'Module members' is by individual procedure and property plus as groups of lines: all declaration lines; all Module-level Attributes; for Access Forms and Reports, all 'control and UI layout' lines

For an individual Module, the diff analysis shows:

- Which 'Module members' † have been added, deleted, updated, renamed and from that you can drill down to:
  - For added or deleted Module members, view the code of the member
  - For updated members, view the code of the two versions compared … see 'Code compare', below … additionally, it is possible to show or hide differences due to line numbers and/or whitespace (only whitespace at the start / end of a line) using the options in the View button's drop-down menu

† breakdown of 'Module members' is by individual procedure and property plus as groups of lines: all declaration lines; all Module-level Attributes; for Access Forms and Reports, all 'control and UI layout' lines

Code compare

Two versions of code are shown and compared at a character-by-character basis and differences are highlighted:

- Green - for code that has been added (will only appear in the upper box representing the version of the code in the VBE that is being 'compared from')
- Red - for code that has been removed (will only appear in the lower box representing the version of the code in the export / restore point that is being 'compared to')
- Yellow - for modifications other than just a change of case ... i.e. a combination of a deletion and an insertion (so can appear in both the upper and lower boxes)
- Blue - for change of case only (so can appear in the upper and lower text boxes)

Version Control

Available in the Project window context menu (when an individual Module name is clicked, menu items for importing / exporting an individual Module will be shown; when the Project name is clicked, menu items for importing / exporting the entire Project will be shown).

Version Control supports the export and subsequent import of one Module or all Modules in a Project. Additionally, when the export or import is for a Project (not an individual Module) then Project References and RibbonX can be included (see the 'Scope' section within the 'Other Version Control related information' section, below, for further information).

Settings for Version Control are available in the 'Version Control' tab in the Settings.

Version Control comes in two 'types' – Plain and Full

- Plain - exports the Module or Modules to a specific folder. When you perform an export, any existing export files in the same folder will be deleted†. Use this option if using a Version Control system e.g. Git (but note that VBE_Extras' Version Control does not integrate directly with Git or any other version control tool).
- Full - exports the Module or Modules plus an XML file to a 'timestamped' folder so creating a new folder for each export. Use this option if you want to export / import on your local device and / or want to have multiple exports available. The XML file will include the date/time of the export and, optionally, a 'reason' (that you specify during the export process) and your username. When performing an import, this information will be shown to you to aid in selecting the correct export to be imported.

† other than UserForm .frx files if the 'Export UserForm .frx files' setting is 'Ask' and you turn down the export of them … see section 'Relevant to host applications that support UserForms' below for more info.

Version Control allows export / import to and from one of 3 specific locations

- Project folder - always exports to and imports from the same folder that the VBA Project (e.g. the Excel Workbook, the Access Database etc) is in.
- Project sub-folder - always exports to and imports from a sub-folder of the of the folder that the VBA Project (e.g. the Excel Workbook, the Access Database etc) is in - that folder will be named the same as the Excel Workbook, the Access Database etc.
- Specific folder - always exports to the same folder for all VBA Projects but allows you to set (and change) that folder each time you perform an export or import.

If using the Full type of Version Control, you can also select to import from sub-folders of the Project folder / Project sub-folder / Specific folder.

In Outlook, "the same folder that the VBA Project is in" means the location of the VBAProject.OTM file. Typically, this is somewhere like: "C:\Users\yourusername\AppData\Roaming\Microsoft\Outlook". For more, see https://learn.microsoft.com/en-us/outlook/troubleshoot/deployment/manage-distribute-outlook-vba-project.

When importing a Module

You have two options:

- 'Import this Module' … allows you to select, and import, a previously exported version of the same Module … before the import takes place, you will always be shown a report of the differences between the current and the previously exported version of the same Module (see Diff analysis for details)
- 'Import new Module' … allows you to select, and import, a previously exported Module that is now no longer in the active Project … before the import takes place, you will always be shown the text of the previously exported Module

For both options, you can select the version of the Module to import from any previous exports of just this one Module and from any previous exports of the entire Project that included this Module.

When importing a Project

You have two options:

- 'Import Project – only add/update Modules' … as it says, only adds Modules that are include in the export but are not in the active Project plus updates Modules that are included in the export but are different in the active Project
- 'Import Project – add/update/remove Modules' … as well as doing the same as in the above bullet, if a Module is not in the export but is in the active Project then it will remove that Module from the active Project

Whichever of these two options you use, you will always be shown a report of the differences between the export and the active Project so that you can make a final decision as to whether to import or not. The differences report is initially shown at the Project level showing the Modules that will be added, deleted or updated and from that you can drill down to the Module level showing the Module-members that will be added, deleted, updated or renamed (see Diff analysis for details).

Other Version Control related information

Scope

The scope of what is exported and imported is as follows:

- In host applications other than Access:
  - VBA code … in all types of Modules
  - VBA attributes … in Standard, Class and UserForm Modules
  - UI elements in UserForms
  - Project References † are exported and (optionally) imported ONLY WHEN exporting / importing the entire Project (not when exporting / importing a single Module)
  - RibbonX 'customUI' file(s) are (optionally) exported and compared-on-import ‡ ONLY WHEN exporting / importing the entire Project (not when exporting / importing a single Module) - not in Outlook or Access
- In Access, in addition to the above:
  - UI elements in Forms and Reports
  - VBA attributes in Forms and Reports

† Project References (sometimes called External References) are the References that a VBA Project holds to external Type Libraries, other VBA Projects, and Templates (e.g. in Word) … these can be seen in the VBE's 'References' dialog available via Tools > References or in VBE_Extras's Update Project References dialog.

‡ RibbonX 'customUI' file(s) cannot be imported but will be included in the 'report of differences' if any differences are found

Other information

See Appendix 6 – relating to Version Control and Restore for sections on:

- Relevant to host applications that support UserForms
- Document Modules in host applications other than Access
- Access-specific
- Case-only Module changes
- Other info

Replace all Modules in this Project

Only available in Excel and Word.

Available in the Project window context menu (when the Project name is clicked).

Replaces all Modules (and so all VBA code and UserForm designers) in the active Project with that from another (the 'source') Project. The active and source Projects must be contained in the same host application. The file (Document / Workbook / Template / AddIn) that contains the source Project must be open and both Projects must be unlocked.

The active Project must contain Document Modules with matching names for each of those in the source Project - that is:
- For Excel:
  - There must be a matching Worksheet with the same name in the active Project for every Worksheet in the source Project
  - There must be a matching Chart with the same name in the active Project for every Chart in the source Project, if any
  - The name of the ThisWorkbook Module in the active Project must match the name of the ThisWorkbook Module in the source Project
- For Word:
  - The name of the ThisDocument Module in the active Project must match the name of the ThisDocument Module in the source Project

The active Project's name will be updated to match that of the source Project. Attributes in standard, Class and UserForm Modules will be updated but those in Document Modules will not be.

The active Project's description, Conditional Compilation Args, password and references to other VBA Projects or Libraries are not updated.

Note for ActiveX Controls and other objects embedded in Worksheets (Excel) / the Document (Word): the code relating to the ActiveX Control (eg Event handlers) will be replaced along with all other code. However, the ActiveX Control itself will not be replaced as it exists in the 'UI' and not in a Module.

Background

The main aim of the Replace all Modules command is to resolve the problem whereby VBA code and data reside in the same file and users need to continue to use the 'live' file and update the data it contains while at the same time the developer needs to update the code. Typically, the data and the code will then be out of sync (i.e. there will be one file with the latest data and another with the updated code) and an exercise takes place at some point to copy the new code over to the file with the correct data … depending on the number of Modules involved, this can be a laborious and error-prone process. Using the Replace all Modules command, this can be completed with a couple of clicks. It should be noted that while this command will copy any Modules from one file to another (noting the provisos above) it is really designed to copy an updated version of the same code from one file to another.

Before anyone reading this contacts me to say that there is a better solution to the described problem (use an AddIn) … I know, I know! However, this command helps anyway:

1. My experience of being asked to fix or update code developed by others that is already in the same file that the data is in drove me to write this command for VBE_Extras. With this command: the customer can send you a version of the file in which the data and code reside; the customer can continue to update the data in their version of the file while you spend time updating the code in your version; you agree a time for the code to be implemented; the customer sends you the latest version of the file (i.e. with the latest data but with now out-of-date code) and freezes making updates; in seconds you update the code in the latest version of the file and get it back to the customer.
2. Occasionally a customer will not want to share 'actual' data with a developer if, for example, it is commercially sensitive. With the Replace all Modules command, the customer can send you a file with dummy data and the real code. You can update the code and send the file to the customer. The customer, having installed VBE_Extras and being shown how to access the VBE, can then import your code with a couple of clicks.

Clean this Project

Available in the Project window context menu (when the Project name is clicked).

Cleans the 'junk code' that builds-up over time when developing macro-enabled files. This 'junk code' can lead to strange problems such as:

- Projects crashing when you open them * or when you click on 'Enable Content' (in the gold banner in the UI), or when you first run the Project's code
- Compile-time or run-time errors for code that is clearly valid, for example "Invalid outside procedure" for something that is clearly within a procedure; "User-defined type not defined" on a VBA reserved-word; "ByRef argument type mismatch" when an argument and the parameter are clearly of the same type

* if you find you cannot even open a Project (eg Excel Workbook, Word Document etc) in order to run the 'Clean this Project' command as the host app force-closes during the opening process, try to: close the Project; rename the file (to anything different); open the file without clicking 'Enable Content'; go to the VBE; in the main VBE menu, select Debug > Compile; save then close the Project; rename the file back to the original name; open the file (click 'Enable Content' if it appears), go to the VBE … at this point, you may find that any problems you were experiencing have been resolved … but if not, then run the 'Clean this Project' command.

How does 'Clean this Project' work?

'Clean this Project' performs a number of steps:

- It performs a full export of all of your VBA code across all Modules of the Project (the same as is done for a Version Control 'Project export') to a temporary folder
- It removes all of the Standard, Class, UserForm and, if running in Access, Document Modules (Forms and Reports) from the Project

- If running in any host app other than Access (as Document Modules cannot be removed from these hosts via the VBE), the code of each Document Module is deleted
- The code is compiled (as there is no code, this effectively deletes all of the previously compiled code including any 'junk code')
- It performs a full import of all of the VBA code that was previously exported (again, the same as is done for a Version Control 'Project import') from the temporary folder

View RibbonX

Only available in Excel, Word and PowerPoint.

Available in the Project window context menu (when the Project name is clicked).

View the RibbonX (Ribbon XML) file(s) stored within the Workbook / Document / Presentation / AddIn. Note that only certain types of file support RibbonX (for example, it cannot be extracted from non-XML files such as .xls or .doc and it cannot be extracted from Excel binary files .xlsb).

Update Project References

Available in the Project window context menu (when the Project name is clicked).

This is an equivalent of the VBE's own 'References' dialog (i.e. that you access via Tools > References) with the advantages of:

- You can search/filter (including "?" and "*" wildcards … see Appendix 8 – wildcards in 'filter text boxes') by name, filename or GUID for specific Type Libraries and Projects
- View both Type Libraries and Projects or just one-or-the-other
- Retains a list of recently added Type Libraries so that you can quickly re-add them (to the same or to a different Project)
- Displays the Version and, optionally, the Flags, Language ID, Resource ID and GUID for each Type Library
- Right-clicking on a Type Library in either the 'Current References' or 'Available References' list:
    - Displays a Type Libraries 'program' name … see 'Why is knowing the 'program' name useful?', below
    - Allows you to copy the name, path or GUID of the Type Library
    - Allows you to copy the 'As <Ref>' code comment for Special binding
    - Allows you to 'explore' the members of the Type Library before (or after) you have added a reference to it … see Type Library explorer

Why is knowing the 'program' name useful?

When using early binding, this is the name you use in VBA code to refer to the library which, when followed by a '.', shows the names of all of the available 'types' that the library provides in the VBE's Intellisense window.

For example, the 'program' name for the 'Microsoft Scripting Runtime' is 'Scripting' … if you add 'Microsoft Scripting Runtime' as a reference and then in VBA code type 'Scripting' followed by a '.', you will see the names of the available 'types' (actually classes in this case) that the 'Microsoft Scripting Runtime' library provides.

While the 'program' name is often similar to the library name or filename, it does not have to be - for example, if you didn't know from other sources, working out that the 'program' name for the 'Windows Script Host Object Model' library (filename 'wshom.ocx') is 'IWshRuntimeLibrary' could take you some time (and the VBE provides no simple way to find it … other than trawling through thousands of entries in the Intellisense window).

Type Library explorer

Explore the members of a Type Library before (or after) you have added a reference to it.

By default, the members are displayed in a parent / child 'tree view' such that child members only appear within their parent. However, this is not how the VBE displays members of Modules or Enumerations … the VBE shows them in a 'flat hierarchy' as though they were also 'top level' members. If you prefer the VBE 'flat hierarchy' then you can enable this in the 'TypeLibs' tab of the Settings.

Hidden members of the Type Library are shown if you have 'hidden members' being shown … see Show / hide hidden members for more.

References of this UserForm

Available in the UserForm designer window context menu.

Same as References for code at cursor but for the UserForm clicked in the UserForm designer window.

Declarations in this UserForm

Available in the UserForm designer window context menu.

Same as List declarations in this Module but for the UserForm clicked in the UserForm designer window.

Rename this UserForm

Available in the UserForm designer window context menu.

Same as Rename code at cursor but for the UserForm clicked in the UserForm designer window.

References for Control

Available in the UserForm Control window context menu.

Same as References for code at cursor but for the Control clicked in the UserForm designer window.

Rename Control

Available in the UserForm Control window context menu.

Same as Rename code at cursor but for the Control clicked in the UserForm designer window.

History - Back / Forward

Available in the Toolbar (when shown) *and by using the Ctrl + - (that is Ctrl and minus) and Ctrl + Shift + - (that is Ctrl and Shift and minus) keyboard shortcut*.

Allows you to move the cursor back to previously visited code locations. Having moved back, you can then move the cursor forward to subsequent locations.

Show History

Available in the Toolbar (when shown) *and by using the Alt + - (that is Alt and minus) keyboard shortcut*.

Shows a drop-down of recently visited code locations (most recent at the top). The current location within the history 'stack' is shown within '>' and '<'. You can then select a location from the drop-down to move the cursor to that previously / subsequently visited location.

If you 'go back' and then carry out an action that causes a new code location to be added to the history then the 'forward' stack is removed.

Project Picker / Module Picker

Use of the Project and Module Pickers requires that the 'Module and Project Pickers are enabled' check box (in the 'Shortcut Keys' tab in the Settings dialog) is checked. The default keyboard shortcuts are Ctrl+Tab for the Module Picker and Ctrl+1 (in the number row, not the number pad) for the Project Picker.

The pickers will be familiar (ish) to users of Visual Studio …

The Project Picker shows a dialog listing the Projects loaded in the VBE in the order they were last accessed, or alphabetically for those never accessed (Projects that are locked are greyed-out). Very handy for going back-and-forth between the two previously active Projects.

The Module Picker shows a dialog listing the Modules (document, standard, Class or UserForm) of the active Project in the order they were last accessed, or alphabetically for those never accessed. Very handy for going back-and-forth between the two previously active Modules. UserForm 'designers' (highlighted in the Picker) are treated as though they are separate Modules so that you can easily navigate between the UserForm code and the UserForm designer.

For either Picker, a single press-and-release of the keyboard shortcut will change the active Project / Module to the previously active Project / Module. You can then continually cycle between the currently active Project / Module and the previously active Project / Module with a further press-and-release of the keyboard shortcut.

Or press the keyboard shortcut then hold down the Ctrl key and use either subsequent presses of the main key (that is, by default, the Tab key for the Module Picker and the 1 key for the Project Picker) or the up/down arrow keys to move between Projects / Modules … release the Ctrl key when you have selected the Project / Module you want to activate.

If the Project Picker is showing, a press of the right arrow key will take you to the Module Picker for the selected Project; if the Module Picker is showing, a press of the left arrow key will take you to the Project Picker.

Both the Project and Module Pickers remember the order that Projects / Modules were last accessed (the Module Picker remembers the order for each individual Project) so if you were working on one Project, change to another and then come back to the original Project, the Module Picker will show the Modules in the correct order.

AutoText

AutoText is added by pressing the Tab, Enter or Return key when the cursor immediately follows the 'AutoText' text that is to be converted. There must be no code (though there can be code-comments) following the cursor position. AutoText will not be added:

- Within comments or literal values
- In lines with multiple statements (delimited with colons)
- In lines ending with (or following from) continuation characters
- In lines with line numbers

Options for AutoText are in the 'AutoText' tab of the Settings dialog (AutoText is switched off by default).

AutoText using '?'

Within a Procedure / Property …

- A solo question mark - the question mark will be replaced with 'Debug.Print ' (trailing space is intentional) … hold down Shift to include 'Now,'
- A question mark which immediately follows:
    - The name of one or more in-scope variables, statics, constants, Type elements, Properties or Functions (for Properties and Functions, only where they take no parameters or all parameters are Optional or ParamArray) and which has a 'primitive' value type – the name and question mark will be replaced with 'Debug.Print "name = " & name & """ … hold down Shift to include 'Now,'
    - The name of an in-scope variable that has a type of a Type – you will be prompted to select an element of the Type then this will be processed as above
    - The name of an in-scope variable that has a type of a Class (defined in your code) – you will be prompted to select a member of the Class (only those that: have the correct access modifier; have/return a 'primitive' value type; and, if a Function or Property Get, those that have no required parameters) then this will be processed as above

- The name of an in-scope variable that has a type of a Class or Interface (defined in a referenced Type Library) – you will be prompted to select a member of the Class / Interface (only those that: have the correct access modifier; have/return a 'primitive' value type; and, if a Function or Property Get, those that have no required parameters) then this will be processed as above … note that AutoText using '?' also works with Special binding
- Anything else – AutoText takes a 'user knows best' approach and processes as per above (for example if you type Application.Name? then press Tab/Enter/Return this will be replaced with Debug.Print "Application.Name = """ & Application.Name & """"

AutoText using '='

A solo equals sign when in a Function or Property Get – the name of the Function or Property Get will be inserted before the equals sign along with 'Set' if the Function or Property Get is returning a non-primitive type.

AutoText using '!'

While the following AutoTexts are referred to as "using '!'", as of release 1.5.4.0, the use of '!' is optional, controlled by the 'AutoText trigger' dropdown in the 'AutoText' tab of the Settings dialog. However, all the 'From' AutoTexts in the tables below are shown with the '!' present.

In the following tables:
- ?? represents a condition (e.g. in an 'If … Then' statement)
- $$ represents the name of a variable or other declaration (e.g. in 'With' statement)
- ## represents a number

Note that:
- The 'From' AutoText may only be preceded by whitespace and followed by whitespace and/or code-comments
- The supplied ??, $$ and ## are not checked by VBE_Extras at the time the AutoText is converted to code (other than for Select Case … End Select blocks when the supplied $$ is a variable with a value type of an Enum) but will, obviously, be checked by the VBE, at compile-time or at run-time, in the same way as any other code
- All of the 'From' AutoTexts are case-insensitive

Anywhere in a Module

From	To
#if! / #if ?? then!	An #If Then … #End If block
#ife! / #ifelse! #if ?? thene! #if ?? thenelse!	An #If Then … #Else … #End If block
#elseif ?? thene! #elseif ?? thenelse!	Inserts a further #Else within the #If Then … #End If block

In the declaration lines of a Module

From	To
opm!	Option Private Module
oe!	Option Explicit
ocb!	Option Compare Binary
oct!	Option Compare Text
ocd!	Option Compare Database *
ob0! / ob1!	Option Base 0 / Option Base 1
enum! / enum $$!	An Enum … End Enum block
type! / type $$!	A Type … End Type block

* = in Access only

Within a Procedure / Property

From	To
For $$ = ## To ##!	A For … Next block

For $$ = ## To ## Step ##!	
for $$ ## ## ##!	A For … Next block • The $$ must be specified and is the For loop 'counter' • The first two ## must be specified and are the For loop 'bounds' • The third ## is optional, if specified then it is used as the Step value
forr $$ ## ##!	A For … Next block with a Step of -1 • The $$ must be specified and is the For loop 'counter' • The two ## must be specified and are the For loop 'bounds'
fora $$ $$ ##!	A For … Next block which iterates over an array from LBound to UBound • The first $$ must be specified and is the For loop 'counter' • The second $$ must be specified and is the array • The ## is optional, if specified then it is used as the array dimension
forra $$ $$ ##!	As for a but iterates over an array from UBound to LBound
For Each $$ In <Collection>	A For Each … Next block
fore $$ $$! foreach $$ $$!	A For Each … Next block • The first $$ must be specified and is the For loop 'element' • The second $$ must be specified and is the collection that is being iterated over
with! / with $$!	A With … End With block
if! / if ?? then!	An If Then … End If block
ife! / ifelse! if ?? thene! if ?? thenelse!	An If Then … Else … End If block
elseif ?? thene! elseif ?? thenelse!	Inserts a further Else within the If Then … End If block
do / do ??! do loop while! / do loop while ??! doloopwhile! / doloopwhile ??! dolw! / dolw ??!	A Do … Loop While block
do while! / do while ??! do loop! / do loop ??! dowhile! / dowhile ??! dowl! / dowl ??! dow! / dow ??!	A Do While … Loop block
do loop until! / do loop until ??! doloopuntil! / doloopuntil ??! dolu! / dolu ??!	A Do … Loop Until block
do until! / do until ??! dountil! / dountil ??! doul! / doul ??! dou! / dou ??!	A Do Until … Loop block
while! / while $$!	A While … Wend block
select! / select $$! select case! / select case $$!	A Select Case … End Select block • If $$ is a variable with a value type of an Enum (defined in VBA code or defined in a Type Library) then the Select Case … End Select block will include a Case line for each Enum member
$$ ++ $$ --	Converted to $$ = $$ + 1 Converted to $$ = $$ - 1
$$ += $$ -= $$*= $$/= $$\= $$^= $$&=	Converted to $$ = $$ + Converted to $$ = $$ - Converted to $$ = $$ * Converted to $$ = $$ / Converted to $$ = $$ \ Converted to $$ = $$ ^ Converted to $$ = $$ &

For all of the AutoTexts that add a 'block' (eg a For … Next block, or an If Then … End If block), when using AutoText without an explicit "!", the code of the Procedure / Property will be parsed to determine when a closing statement for the block already

exists and, if so, no closing statement will be added … this is to prevent, for example, a duplicate closing End If statement being added if you were to press the Enter key following an If Then statement).

Auto-indent following underscore

Following pressing the Enter or Return key (and, therefore, requires the AutoText trigger to include the Enter and Return keys) at the end of a physical line which ends with the line continuation character sequence (i.e. a space followed by an underscore), will auto-indent the next physical line (the size of the indent being based on the 'Indent length' and 'Continued lines' settings in the 'Indent/Split' tab of the Settings dialog).

Pressing the Enter or Return key at the end of a subsequent physical line (which is part of the same logical line):

- Which also ends with the line continuation character sequence, will main the indent for the next physical line
- Which does not end with the line continuation character sequence, will outdent the next physical line by the amount of the original indent

Auto closing pairs

As you type (or delete) one member of a 'pair', VBE_Extras will add (or delete) the partner of that pair. Pairs are:

- Parentheses: ( )
- Speech marks: " "
- Square brackets: [ ]
- Braces: { }
- Hash symbol (number sign): # #

Note that the pair will only be auto closed if typing at the end of a physical line or, when typing within a line (i.e. there are one or more characters to the right of the insertion point), if the character immediately to the right of the insertion point is a space character or is itself the closing half of a pair. Auto closing pairs does not work in code comments.

Select which pairs will be auto closed in the 'AutoText' tab of the Settings dialog (all auto closing pairs are switched off by default).

A double-press of the Shift key before typing will temporarily (for 5 seconds) suspend the adding or deleting of the partner of the pair.

Holding down the Shift key while pressing the Backspace key will prevent the partner of the pair from being deleted.

Specific to hash symbol (number sign)

The Settings do not allow hash symbol to be auto closed within a string literal: a pair of hash symbols is used to indicate a date literal and you cannot place a date literal within a string literal.

Additionally, hash symbol will not be auto closed if:

- It is typed immediately following a digit ... it is assumed that the # is being used to indicate a literal double value
- It is typed immediately following a character allowed in a VBA declaration name ... it is assumed that the # is being used to indicate that the declaration has a Double value type
- It is typed at the very start of a line or only following whitespace on the line ... it is assumed that the # is being used as part of a conditional compilation statement

Alternative VBE window navigation keys

Makes some of the built-in keyboard shortcuts used to navigate between certain VBE windows more logical:

- When viewing a UserForm, F7 alternates the view between the code AND THE designer
- F2 navigates both to AND FROM the Object Browser
- F4 navigates both to AND FROM the Properties window

- Ctrl+G navigates both to AND FROM the Immediate window
- Ctrl+R navigates both to AND FROM the Project window
- Ctrl+L both shows AND CLOSES the "Call Stack" window (only when stepping through code or when code execution is paused)

... note that if you have allocated any of the above keyboard shortcuts to other VBE_Extras commands (or to commands provided by other VBE AddIns) then they may continue to trigger that other command instead.

To enable this functionality, ensure that 'Alternative VBE window navigation keys' is checked in the 'Other' tab of the Settings dialog.

Store / restore

Store / restore Conditional Compilation Arguments

If the 'Store / restore Conditional Compilation Arguments' check box (in the 'Other' tab of the Settings dialog) is checked then Conditional Compilation Arguments that have been updated (either automatically or manually, using the relevant menu items … see Conditional Compilation Arguments) for a VBA Project will be stored when the VBA Project is closed, and restored when the VBA Project is re-opened. This means that you don't have to remember to update CCAs every time you open a VBA Project, rather only when you change the CCA names or values in the VBE Project Properties window.

By default, this functionality only works when the opened VBA Project is not locked (i.e. not password-protected) ... to also restore when a VBA Project is unlocked, also check the 'Restore when a Project is unlocked' check box (in the same tab of the Settings).

Store / restore cursor location

If the 'Store / restore cursor location' check box (in the 'Other' tab of the Settings dialog) is checked then the location of the cursor will be stored when you close a VBA Project. When you re-open the same VBA Project and open/activate the VBE, VBE_Extras will show you a dialog offering to restore your cursor to that location.

By default, this functionality only works when the opened VBA Project is not locked (i.e. not password-protected) ... to also restore when a VBA Project is unlocked, also check the 'Restore when a Project is unlocked' check box (in the same tab of the Settings).

Clearly, the cursor location can only be restored if the saved location exists in the re-opened VBA Project. For example, if you save a VBA Project, add a Module, move the cursor to that Module and then close the VBA Project without saving then the saved location will not exist within the VBA Project when it is re-opened. In this case, you will not see the usual dialog offering to restore your cursor location.

Additionally, note that the cursor location can only be stored if VBE_Extras is loaded in the VBE. If you were storing the cursor location and then unload VBE_Extras, no further cursor locations will be stored. If you then re-load VBE_Extras, when opening VBA Projects, then the restored location may be out-of-date.

Store / restore … limitations

- The 'Store / restore' functionality only works when VBA code is not running
- The stored information is not saved with the file (i.e. it is not saved in the Workbook / Database / Document etc), rather it is saved locally per-device … as such:
    - If you work on the same file across multiple devices, the stored information will be discrete on each device
    - If you change the name of a file, move a file, or change the name of any folder in the file's path, the stored information for that file will not be found, and so will not be restored

F1 to view Windows API web pages

In the 'plain' VBE, the F1 key is used to open the Microsoft documentation web page for the keyword at the cursor.

VBE_Extras extends this functionality to open the Microsoft documentation web page for Windows API Functions (i.e. that you have declared using the 'Declare' keyword), Types, Enums and Consts … as of this functionality first being added to VBE_Extras in August 2025, VBE_Extras is aware of the appropriate web pages for:

- 12,052 Functions (counting Functions that include both an ANSI and Wide variant as 1 … if counted as 2 discrete Functions then that would be many more)
- 3,315 Types
- 586 Enums
- 10,858 Consts

To use this functionality, ensure that 'F1 shows help for Windows API declarations' is checked in the 'Other' tab of the Settings dialog, then place the cursor on:

- The name of a Function, Type, Enum, Enum member or Const used in the Windows API … then press F1 … if VBE_Extras "knows" the appropriate web page then it will be opened for you in your default browser
- The alias name of a Function (i.e. that is in speech marks following the keyword "Alias") … then press F1 … does the same as in the above bullet
- The library name of a Function (i.e. that is in speech marks following the keyword "Lib") … then press F1 … if VBE_Extras "knows" the appropriate web page for the library "header" that the Function is declared in then it will be opened for you in your default browser. Why the web page for the library "header" and not the web page for the library itself? Because that is how Windows API functionality is organised … each library is made up of 1 or more headers (with each header containing Functions, Types, Enums and Consts) and each header has a web page. Typically, these web pages contain lists (and brief descriptions) of each Function, Type and Enum that the header contains so this can be very helpful for discovering related functionality within the Windows API

Note that while Windows API Functions that you 'declare' must use the correct name (i.e. the Alias name must exactly match the name or index number of the Windows API), any Types, Enums, Enum members and Consts do not have to use the correct name (they just have to use the correct structure … if a Type … or the correct value(s) … if an Enum/member or Const). However, if you do not use the *exactly correct* name for Types, Enum/members and Consts then VBE_Extras will not be able to identify that it is one defined in the Windows API and so will not be able to open the web page.

Note also that the web page opened will always be the English language ("en-us") web page.

Toolbar

VBE_Extras includes a Toolbar with a set of buttons controlling multiple VBE_Extras commands.

The visibility and exact location of the Toolbar are not controlled by VBE_Extras, rather by the VBE itself. To make the Toolbar visible or to hide it, in the VBE select View > Toolbars > VBE_Extras_Toolbar. To move the Toolbar, hover the mouse pointer over the dots on the far-left side of the Toolbar until you see the 4-pointed mouse pointer then drag the Toolbar to your desired location.

To change the buttons that are included in the Toolbar, either:
- Select Extras > Settings … then select the UI tab and click on 'Customise Toolbar' (this button also includes an option to reset the set of buttons in the Toolbar back to the defaults)
- Click on the 'Toolbar Options' arrow (at the far-right side of the Toolbar) then "Add or Remove Buttons" then "Customize..."

… which takes you to the "Customise Toolbar" dialog.

When customising the Toolbar, depending on the host application you are using, you may notice that buttons can be added to the Toolbar for commands that are not available for the host application (the names of such commands will be greyed-out). The reason for this is that the same Toolbar with the same buttons/commands is used in VBE_Extras across all host applications … if buttons/commands that are not available in your currently-being-used host application were omitted and then you saved the Toolbar, those buttons/commands would be removed ... then when you next used VBE_Extras in a host application that does allow those buttons/commands, they would be missing. Hence all buttons/commands are shown in the "Customise Toolbar" dialog but only the relevant buttons/commands for your host application are shown in the actual Toolbar.

In the Toolbar, the buttons for the following commands are 'dynamic':

- History back / forward / show - enabled/disabled based on whether there is any appropriate 'history' to go to … see History - Back / Forward and Show History
- Run 'Sub To Run' and Clear 'Sub To Run' - enabled/disabled when there is a Sub To Run set … see Set Sub to run / Run Sub to run / Clear Sub to run
- Run Again - enabled/disabled when there is a Sub to Run Again set … see Run Again
- Show / hide hidden members - is 'up' when hidden members are being hidden; is 'pressed-down' when hidden members are being shown … see Show / hide hidden members

Mouse-click commands

Mouse clicks (combined with Ctrl or Alt) on declarations, references and literal values perform specific commands:

	While holding down Ctrl …	While holding down Alt …
… click on a declaration	See a list of all references for that declaration (the same as References for code at cursor) allowing you to jump to any of those references	Highlight the on-screen declaration and references for a specific declaration (the same as Highlight)
… click on a reference	Jump to the declaration (the same as Go to declaration for code at cursor)	
… click on a literal value	See a list of all matching literal values (the same as Matches for literal value at cursor) allowing you to jump to any of those values	n/a
… click on a 'word' *	n/a	Highlight the on-screen matches for that 'word' * (the same as Highlight)

Note that if the entire declaration name / literal value / 'word' is selected, then you will have to double-click it to perform the mouse-click command.

* the 'word' can be a VBA keyword or any other text that is not in a string literal or in a code comment

Whether the Mouse-click commands are disabled, enabled, only enabled for the 'Ctrl' commands, or only enabled for the 'Alt' commands can controlled in the Settings dialog.

VBE_Extras Helper

Only available in Excel.

VBE_Extras Helper is an Excel AddIn that supports working with Shapes, Tables and Names in the Excel UI. VBE_Extras Helper must be installed separately from VBE_Extras itself.

When using VBE_Extras Helper, if you receive a message "For this command to work, VBE_Extras must be installed and loaded in the VBE" then either:

- VBE_Extras is not installed at all - see Installation , or
- VBE_Extras is not loaded in the VBE … to fix, in the VBE main menu (not in the host app's 'ribbon' menu): Add-Ins > Add-In Manager … in the list of 'Available Add-Ins' highlight VBE_Extras and put a tick/check in both the 'Loaded/Unloaded' and 'Load on Startup' checkboxes

The following commands can be accessed in the Excel UI either in the 'Add-ins' tab of the ribbon menu or via the right-click context menu for Shapes / Tables:

Shapes

Uses of Shape's name

For the selected Shape, list all String literals matching the Shape's name property in the relevant Workbook or AddIn:
- First, the VBA Project in the Workbook that the Shape is within is searched
- Second, if there are no matches for the Shape's name in the VBA Project in the Workbook AND if the active VBA Project is in an AddIn, then search within the AddIn

This command is not 'code-context sensitive' ... all matching String literals will be returned:
- Whether they refer to a Shape name or not, and whether they refer to a Shape in the right Worksheet or Chart sheet or not
- Matches are not case-sensitive

This command requires the VBE for the host application to have been opened at least once during this session of the host application, and for VBE_Extras to be loaded.

For the reverse action, see Go to Shape with this name or that calls this Procedure.

View / Copy Shape's name

View and copy the selected Shape's Name property.

Go to OnAction

For the selected Shape, go to the procedure that is the target of the Shapes' OnAction property (as set using 'Assign Macro').

This command requires the VBE for the host application to have been opened at least once during this session of the host application, and for VBE_Extras to be loaded.

For the reverse action, see Go to Shape with this name or that calls this Procedure.

View / Copy OnAction

View and copy the selected Shape's OnAction property (as set using 'Assign Macro').

Tables

Uses of Table's name

The same as Uses of Shape's name but for Tables.

For the reverse action, see Go to Table with this name

View / Copy Table's name

View and copy the selected Tables Name property.

Names

Uses of Name's name

The same as Uses of Shape's name but for Names.

For the reverse action, see Go to Name with this name

View / Copy Name's name

View and copy the selected Name's name property.

MS Docs

Via the Info for code at cursor and Show All Members commands, VBE_Extras will open the MS Docs web pages for objects and members in the following libraries:

- Visual Basic For Applications
- Microsoft Office Object Library
- Microsoft Forms Object Library
- Microsoft Excel Object Library
- Microsoft Word Object Library
- Microsoft PowerPoint Object Library
- Microsoft Outlook Object Library
- Microsoft Access Object Library
- Microsoft Project Object Library
- Microsoft Publisher Object Library
- Microsoft Visio Object Library
- Microsoft Office Access database engine Object Library
- Microsoft ActiveX Data Objects Library
- Microsoft Visual Basic for Applications Extensibility
- Microsoft Scripting Runtime

The ability to open the web pages is available when using early binding or when using Special binding.

Note that:

- The web page opened is always the US English page
- Identification of the correct web page is far from perfect ... the irregularities in the URLs (typos, mis-categorisations) for the documentation and missing documentation (especially for recently added objects/members, for example Excel.Range.InsertPictureInCell and Excel.Application.FormatStaleValues) mean that this functionality inevitably contains errors and omissions
- There is limited documentation available for VBA.RegExp

Known problems

Following renaming a Class Module that is used as an Interface, when you try to compile the Project, occasionally the VBE will show a warning dialog with a message such as "User-defined type not defined" ... the VBE will not jump to a specific line for the error (as there is no error). This can be resolved by either:

- Find the 'Implements' statement in the Class that implements the Interface, change the name of the Interface (to something invalid), compile the Project (to create a genuine error), then change the name of the Interface back to what it should be, compile again … error cleared
- Ignore the error and just run the code in your Project … it will run despite the 'error' and the error will be cleared

VBE_Extras is suddenly missing from the VBE:

- Check that VBE_Extras is loaded … see Check the AddIn Manager
- Close the host application. If the host application supports multiple instances of itself running at once, close those as well. Then use Task Manager (via Ctrl+Alt+Delete) to check for and kill any hidden ('zombie') instances of the host application. Re-start the host application and re-open the VBE.

Reporting a problem

If you have VBE_Extras running on your device, please use the Report a bug menu option - this will help you to clarify your problem report and ensure the right information is provided by you so that I can assist with the problem. If you are reporting a problem that relates to specific VBA code, please try to include a Minimal, Reproducible Example (see

https://stackoverflow.com/help/minimal-reproducible-example) when making contact so that any problem or bug can be reproduced. Without being able to reproduce your problem or bug, it is unlikely that a fix can be developed.

If your problem is with getting VBE_Extras to install or run on your device, then please see the Appendix 1 – Installation troubleshooting section.

Performance

To improve the performance of VBE_Extras:

- Save all VBA Projects open in the host application at least once
- Use more / smaller Modules, not less / larger Modules
- Use fully qualified references (see Terminology)
- Use appropriate access modifiers (i.e. don't make everything Public)
- Ensure that Logging (in the Settings) is not set to Verbose … this can have a very significant impact on performance … it is designed to be switched to Verbose briefly while you perform an operation prior to sending a log file to the developer for analysis (Verbose will always be switched off initially when you close the last running VBE host application on your device)

Appendix 1 – Installation troubleshooting

Please first ensure you have correctly followed the Installation instructions. Note that VBE_Extras only works on Windows, versions from Windows 7 onwards. It also requires .NET Framework 4.8 to be present on your device - .NET Framework 4.8 is installed by default on newer versions of Windows 10 (May 2019 onwards) and all versions of Windows 11 – to get the .NET Framework 4.8, see If you need to install or check for .NET Framework 4.8. Obviously VBE_Extras also requires Excel, Word, Outlook †, PowerPoint and/or Access (the 'host applications') to be installed. If your device does not meet these requirements then VBE_Extras cannot be installed.

VBE_Extras is installed on a per-user basis. If your device has multiple users then each user will need to separately install VBE_Extras.

To test whether VBE_Extras is installed correctly, open any one of the host applications, access the VBE (either by pressing Alt + F11, or using the 'Visual Basic' button in the 'Developer' tab of the ribbon) and look for the 'Extras' menu item in the main menu across the top of the VBE window. If you can see 'Extras' and if you can use any of the menu items that it contains, then VBE_Extras has installed correctly. If not:

- Firstly, if any of the Specific problems matches your situation then follow the advice there … if not
- Secondly, follow each of the Troubleshooting steps (the order they are listed in is recommended but not essential) and check to see whether VBE_Extras loads following each one …

† see Appendix 5 – 'new Outlook'

Specific problems

If you install or update VBE_Extras but VBE_Extras is not visible in the VBE (and the 'Extras' menu item is missing)

Follow the Troubleshooting steps starting with Check the AddIn Manager.

If you install or update VBE_Extras but, when opening the VBE, get an error message that "'VBE_Extras' could not be loaded."

The install / update was not completed successfully, likely one or more registry keys was not updated correctly or a file is missing.

To fix, you can either …

Uninstall and re-install VBE_Extras:

- Ensure you are the only user logged on to the device
- Ensure all VBE-enabled applications are closed (including Excel, Access, Word, PowerPoint and Outlook)
- Uninstall VBE_Extras – in Control Panel > Programs and Features (or you can select Settings > Apps) and uninstall VBE_Extras (all versions if multiple are present) … you can choose to not delete your settings if you want them to be available following re-installation
- Go to the VBE_Extras downloads page https://www.thevbahelp.com/vbe-extras-download and download and re-install the latest version of VBE_Extras (see the Installation instructions if required)
- If you still experience the problem, then re-boot your device and try again

Or try an Admin install together with Check the Registry.

If you update to a new version but still see an old version number in the 'About' dialog

The update was not completed successfully, likely one or more registry keys was not updated correctly or a file is missing. To fix, carry out the same steps as for If you install or update VBE_Extras but, when opening the VBE, get an error message that "'VBE_Extras' could not be loaded.", above.

If you need to install or check for .NET Framework 4.8

The VBE_Extras installer will automatically check for .NET Framework 4.8 and inform you if it is not on your device. If the installer shows a dialog informing you that it is missing, then you will need to download it from the Microsoft website.

From time-to-time, Microsoft changes the links to its websites but, at the time of writing, it can be downloaded from https://dotnet.microsoft.com/en-us/download/dotnet-framework/net48 (only the 'runtime' is required to run VBE_Extras, not the 'developer pack') … once downloaded, run the .exe file to install the .NET Framework 4.8.

To check for yourself whether .NET Framework 4.8 is installed, you can either:

- Download and run the .NET Framework 4.8 runtime installer from the link above … if you already have .NET Framework 4.8 installed then you will see a dialog with the text ".NET Framework 4.8 or a later update is already installed on this computer" (or similar text)
- Follow Microsoft's how-to guide (which will allow you to determine the exact version of the .NET Framework that is installed on your device, if any) at https://learn.microsoft.com/en-us/dotnet/framework/migration-guide/how-to-determine-which-versions-are-installed#detect-net-framework-45-and-later-versions

For further details on installation and .NET Framework in general, see https://docs.microsoft.com/en-us/dotnet/framework/install/.

If you receive an 'InvalidCastException' when starting the VBE

This also may be shown as 'System.InvalidCastException'.

There are two most likely causes of this:

1. VBE_Extras is 'clashing' with another VBE AddIn.

   In the VBE, select the Add-Ins > Add-In Manager menu option and untick the 'Loaded/Unloaded' and 'Load on startup' checkboxes for all AddIns (including for VBE_Extras) and OK to close the dialog. Then open the dialog again and tick the 'Loaded/Unloaded' and 'Load on startup' checkboxes for VBE_Extras then OK to close the dialog. If VBE_Extras now loads correctly then it is 'clashing' with one or more other AddIn(s). If you have multiple other AddIns installed (ie as well as VBE_Extras) then likely the clash will be with only 1 of those other AddIns. You may want to experiment with ticking the 'Loaded/Unloaded' and 'Load on startup' checkboxes for each other AddIn one at a time to identify which other AddIn VBE_Extras is clashing with.

2. You have multiple versions of Office installed and have: mixed the bitness (all versions should be either 32 or 64 bit); or have installed in the wrong order (you should install the oldest version of Office first eg 2016 before 2019, before 2021 etc);

or have installed Office 365 together with another version of Office and neither version is installed using MSI (Microsoft Software Installer, aka Windows Installer service)

See https://support.microsoft.com/en-us/office/install-and-use-different-versions-of-office-on-the-same-pc-6ebb44ce-18a3-43f9-a187-b78c513788bf for more info. The only fix for this, if you want to use VBE_Extras, is to uninstall and then re-install the Office versions following the guidance at the link.

Troubleshooting steps

Check the AddIn Manager

In the VBE for any of the host applications, select the Add-Ins > Add-In Manager menu option and look for VBE_Extras in the 'Available Add-Ins' list. If VBE_Extras is present, select it then ensure that 'Loaded/Unloaded' is checked and (if you want VBE_Extras to automatically load when the VBE is opened) that 'Load on Startup' is checked. OK to close the dialog.

If you then get a message that "'VBE_Extras' could not be loaded" (or similar), do you have other VBE AddIns installed? Back in the AddIn Manager dialog, untick the 'Loaded/Unloaded' and 'Load on Startup' checkboxes for any other AddIns and OK to close the dialog. Then open the dialog again and tick the 'Loaded/Unloaded' and 'Load on Startup' checkboxes for VBE_Extras then OK to close the dialog. If VBE_Extras now loads correctly then it is 'clashing' with one or more other AddIn(s). If you have multiple other AddIns installed (i.e. as well as VBE_Extras) then likely the clash will be with only 1 of those other AddIns. You may want to experiment with ticking the 'Loaded/Unloaded' and 'Load on Startup' checkboxes for each other AddIn one at a time to identify which other AddIn VBE_Extras is clashing with.

Re-boot your device

I have seen instances where VBE_Extras (and other VBE AddIns, if any) have disappeared from the AddIn Manager dialog (see Check the AddIn Manager) and they have re-appeared following a re-boot of the device.

Having re-booted your device, if VBE_Extras is not visible in the VBE then Check the AddIn Manager.

Check disabled AddIns

In the host application (not in the VBE), select File > Options > Add-ins. Next to Manage, select Disabled Items then click Go. If VBE_Extras is listed (or if mscoree.dll is listed), then select it then click 'Enable'.

If you found a disabled AddIn, if VBE_Extras is not visible in the VBE then Check the AddIn Manager.

Check the Registry

NOTE THAT CHANGING VALUES IN THE REGISTRY CAN LEAVE YOUR DEVICE UNUSABLE … PROCEED WITH CAUTION AND AT YOUR OWN RISK.

Following this step only makes sense if you have first followed the Check the AddIn Manager step and if VBE_Extras was not visible in the Add-In Manager or could not be loaded following using the Add-In Manager.

Close all host applications (and any other applications you may have installed on your device that you know can use VBA).

Open the Registry Editor (at the Windows 'Start' button, search for 'regedit', or at a command prompt, type 'regedit') and navigate to the following key depending on the bitness of your Microsoft Office installation:

For 32-bit: HKEY_CURRENT_USER\SOFTWARE\Microsoft\VBA\VBE\6.0\Addins\VBE_Extras.Connect
For 64-bit: HKEY_CURRENT_USER\SOFTWARE\Microsoft\VBA\VBE\6.0\Addins64\VBE_Extras.Connect

Within the relevant key, you should see the following entries

Entry	Type	Expected value
(Default)	REG_SZ	(value not set)

Description	REG_SZ	VBE Extras ... extending the VBA editor
FriendlyName	REG_SZ	VBE Extras
LoadBehavior	REG_DWORD	0x00000003 (3)

If any of the entries are missing, you have the option to either manually add them and their values using the Registry Editor, or to uninstall / re-install VBE_Extras and re-installing will add the entries back to the Registry.

If all of the entries are present, then note that only the value for LoadBehavior is critical. If it is not as expected, double-click on LoadBehavior. Enter '3' (without the quotes) as the 'Value data', press OK.

Repair Office

In Control Panel, select 'Programs and Features' (or in Settings, select 'Apps') then find the entry for Microsoft Office (may be listed as Microsoft 365 or a similar name), click Change (or Modify if you are in Settings) then either Quick Repair or Online Repair.

If you do Quick Repair first and it does not help then subsequently try Online Repair. Note that an Online Repair can take a long time.

Temporarily switch off your AntiVirus

NOTE THAT SWITCHING OFF YOUR ANTI-VIRUS OBVIOUSLY CARRIES A RISK AND CAN POTENTIALLY EXPOSE YOUR DEVICE AND YOUR DATA TO VIRUSES AND OTHER MALWARE … PROCEED WITH CAUTION AND AT YOUR OWN RISK.

I suggest you first disconnect from any network / internet before switching off your AV. Then close all host applications, then:
1. Re-open one of the host applications, access the VBE and, if necessary, following the steps in Check the AddIn Manager … if no success, then
2. With your AV still switched off, uninstall and then re-install VBE_Extras then follow step 1

REMEMBER TO SWITCH YOUR ANTI-VIRUS BACK ON WHEN YOU HAVE COMPLETED THIS STEP!

Speak to your IT support

Especially if your device is managed by an IT support team, it is possible that Group Policy settings disables all AddIns and/or VBE AddIns. As a user, you will not be able to change the Group Policy settings. Speak to your IT support team.

Contact me

Before you do, please follow each of the above installation troubleshooting steps. They will resolve 99%+ of all problems - not following all of the steps wastes your time and mine!

You can contact me using the email address on the website https://www.thevbahelp.com/vbe-extras-download, please let me know:
1. The exact text of any error messages.
2. The version of VBE_Extras you are trying to install (shown on the VBE_Extras download page and in the installation dialog while installing) – is always 4 numbers delimited by dots.
3. Your version of Windows.
4. The 'bitness' of Windows (32 bit or 64 bit).
5. Your version of Office … or versions if you're running multiple versions of Office side-by-side.
6. The 'bitness' of Office (32 bit or 64 bit) – note that this is distinct from the 'bitness' of Windows.
7. The host application(s) you have tried to use VBE_Extras in.
8. The 'Office display language' that you use … this is the language that is used in the ribbon, menu items, toolbars etc … it is typically set in: File > Options > Language > Office display language
9. Does VBE_Extras appear in the AddIn Manager dialog (see Check the AddIn Manager)? What message, if any, do you get following ticking the 'Loaded/Unloaded' and 'Load on startup' checkboxes and clicking on OK?
10. The names of any other AddIns installed in the VBE (see Check the AddIn Manager)?
11. Whether this is a new / first installation of VBE_Extras or an update … if the latter, roughly how long have you had VBE_Extras installed on the same device for the same user?

12. Attach a copy of the installation log file (the most recent, if multiple) … this will be in your 'Temp' folder (normally "C:\Users\%username%\AppData\Local\Temp") and called "Setup Log YYYY-MM-DD #NNN.txt" where YYYY-MM-DD is the date of the attempted installation and NNN is an iteration number starting from 001

And, if possible:

13. Whether any other users using the same device already have VBE_Extras installed?
14. The version of the mscoree.dll file on your device … the full path to this file is "C:\Windows\System32\mscoree.dll" … right-click on it then select Properties then Details … the File and Product version will likely be the same, if not, please advise both
15. The name of the anti-virus software you use on your device (eg Microsoft Defender Antivirus or Norton AntiVirus Plus)
16. Also helpful, but not essential, the message shown by the VBScript, below
17. And again helpful, but not essential, is to know all versions of the .NET Framework you may have installed on your device … see https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/how-to-determine-which-versions-are-installed
18. Attach a copy of the VBE_Extras runtime log and exception file(s) … these will only be present on your device if you previously had a successful installation of VBE_Extras … these will be in the "C:\Users\%username%\AppData\Local " folder in sub-folder "VBE_Extras" … optionally Zip and provide the entire "VBE_Extras" folder

VBScript

Create a text file (i.e. a file with a .txt extension) on your Desktop or anywhere else convenient on your device. Add the following text:

```
dim o
set o = CreateObject("VBE_Extras.Connect")
msgbox "OK"
```

Save and close the text file. Change its extension from .txt to .vbs then double-click to run what is now a VBScript file. Make a note of what message, if any, you receive other than "OK" (the message, if any, will be shown in a dialog … you can press Ctrl+C to copy the entire text of the message while the dialog is showing).

Admin install

By default, VBE_Extras is installed per-user. If you need to install for all users on a device then you can try an admin install.

First, locate the full path to the VBE_Extras.dll file which, if you have attempted a per-user install first, will be at "C:/Users/%username%/AppData/Local/Programs/VBE_Extras/VBE_Extras.dll" (where %username% is your username).

Then confirm the 'bitness' of your Office installation (note this is not necessarily the same as the 'bitness' of Windows) … in any Office application, select File > Account > About <AppName> and towards the top of the resulting dialog will be details of the application including either 32-bit or 64-bit.

Open an admin-level Command Prompt window (at the Windows start button, type "cmd" then select "Run as administrator"). In that Command Prompt window, run RegAsm (to register the assembly):

For 32-bit Office, type (or copy/paste) the following (all on one line), replace the path, then press enter:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe
"C:/Users/%username%/AppData/Local/Programs/VBE_Extras/VBE_Extras.dll" /codebase
```

For 64-bit Office, type (or copy/paste) the following (all on one line), replace the path, then press enter:

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\RegAsm.exe
"C:/Users/%username%/AppData/Local/Programs/VBE_Extras/VBE_Extras.dll" /codebase
```

If the resulting message:

- Ends with "Types registered successfully" then the install was successful
- Includes the text "Admin permissions are needed" then you only opened a 'normal' Command Prompt window … try again with an admin-level Command Prompt window
- Includes the text "Unable to locate input assembly" then the path to the VBE_Extras.dll file was not correct … fix the path and try again

Appendix 2 – why Shapes, Tables and Names?

And why only in Excel?

Simple answer, to save time. When I'm coding VBA, I find that I spend three quarters or more of my time doing it in Excel. And Shapes, Tables and Names can be awkward to find … especially if you have a Workbook with many Worksheets each of which has many Shapes, Tables and/or Names. Being able to jump instantly to the relevant object in the user-interface or to jump instantly from the user-interface into the code that refers to the object saves a huge amount of time. And that's why I wrote VBE_Extras … to save time.

Why not other objects in the user-interface? Worksheets and Chart sheets would be obvious … but they're easy to find. Ranges / cells are also obvious but (thinking about Worksheets), every Worksheet has the same references … which Worksheet to jump to for e.g. Range "A1" … and then what about Cells(1, 1) … too much work to code, too little benefit. They're easy to find. No time saved.

And what about all of the other vast array of objects that can be added to the Excel user-interface … Charts, PivotTables etc etc etc? Well, I could add the same functionality, but I tend to find that there will only ever be a small number of each of these in any one Workbook, so they tend to be easy to find … so no time saved.

Appendix 3 – Update history

This table only reflects the versions since 1.0.0.0. Every release includes some level of bug fixes and minor 'tweaks' … that fact is not repeated in the 'notable changes' for every release unless it is a 'bug fix only' release or specific bug numbers are included.

Date	Version	Notable changes
19th Jan 26	1.7.2.0	• Fix case • Open MS Docs pages via 'Show All Members' and 'Info For Code At Cursor' • 'AutoText' for ++, --, +=, -=, *=, /=, \=, ^=, &=
30th Oct 25	1.7.1.2	• Bug fixes only
24th Oct 25	1.7.1.1	• Added more Windows API declarations and examples
14th Oct 25	1.7.1.0	• Consts to Enum / Enum to Consts • Added more Windows API declarations and examples
3rd Oct 25	1.7.0.2	• Added more Windows API declarations and examples
1st Sep 25	1.7.0.1	• F1 to view Windows API web pages now also works on references (not just declarations) • Better handling of custom device "Scale" • Bug fixes including #33
18th Aug 25	1.7.0.0	• Add Windows API declarations • F1 to view Windows API web pages
27th Jun 25	1.6.1.2	• Bug fixes only
9th Jun 25	1.6.1.1	• Bug fixes only
5th Jun 25	1.6.1.0	• Copy as HTML
29th May 25	1.6.0.9	• Bug fixes only
28th May 25	1.6.0.8	• Improved performance in dialogs showing many items • Setting to restrict text filtering in dialogs showing many items
16th May 25	1.6.0.7	• Bug fixes only
9th May 25	1.6.0.6	• Improved parsing of Type Libraries • Updated Show All Members and Type Library Explorer dialogs • Bug fixes including #32
26th Apr 25	1.6.0.5	• Bug fixes only
18th Apr 25	1.6.0.4	• Allow wildcards '?' and '*' in 'filter text boxes'
16th Apr 25	1.6.0.3	• Expand selection / Contract selection
2nd Apr 25	1.6.0.2	• Bug fixes only including #31
28th Mar 25	1.6.0.1	• Bug fixes only including #30
26th Mar 25	1.6.0.0	• Restore points • Undo rename … removed

12th Mar 25	1.5.4.2	• Bug fixes only
9th Mar 25	1.5.4.1	• Bug fixes only including #29
5th Mar 25	1.5.4.0	• Auto closing pairs • Store / restore CCAs and cursor location • Alternative VBE window navigation keys
16th Feb 25	1.5.3.3	• Bug fixes only including #27, #28
23rd Jan 25	1.5.3.2	• Go to last code edit location • Dialogs showing lists of references includes a column for 'Use' • Customisable font size for dialogs
6th Jan 25	1.5.3.1	• Bug fixes only
4th Jan 25	1.5.3.0	• Add parameter / Promote local variable to parameter / Remove parameter / Demote parameter to local variable / Reorder parameters • The buttons included in the Toolbar can now be customised
10th Dec 24	1.5.2.5	• Bug fixes only
25th Oct 24	1.5.2.4	• Bug fixes only including #25
1st Oct 24	1.5.2.3	• Bug fixes only including #24
7th Sep 24	1.5.2.2	• Bug fixes only including #23
31st Aug 24	1.5.2.1	• Compare Module can compare to Modules loaded in the VBE (not just those saved as files) • Compare Project can compare to Projects loaded in the VBE (not just those saved as files) • UI tab in the Settings • UserForm designer context menu
22nd Aug 24	1.5.2.0	• Added 'Command search' command • Added 'Find Setting' capability to the Settings dialog • Compare for Compare Module, Compare Project and Version Control (when importing) is now character-by-character and identifies insertions, deletions and changes separately • Update the VB_Description Attribute in Access Forms and Reports (experimental)
1st Aug 24	1.5.1.3	• Bug fixes only
22nd Jul 24	1.5.1.2	• Bug fixes only
19th Jul 24	1.5.1.1	• GetEnumAsString evaluates expressions used to define the value of Enum members
1st Jul 24	1.5.1.0	• 'Add a "GetEnumAsString()" Function'
24th Jun 24	1.5.0.0	• Can now parse Type Libraries – 'Show All Members', 'Info for code at cursor', 'Qualify reference' and 'AutoText' (for Select…Case) updated • Type Library explorer • Add Enum from Type Library • Special binding
16th May 24	1.4.8.0	• Update "Project References" • Version Control / Compare Project – option to ignore line numbers
3rd May 24	1.4.7.1	• Licensing changes
22nd Apr 24	1.4.7.0	• Version Control – Project references are imported • Version Control – RibbonX is exported, warn on import • Extended Highlight functionality • View RibbonX
8th Apr 24	1.4.6.0	• Line numbers – add/update and remove • Version Control – Project references are exported, warn on import • Compare Project • Orphans commands check for read and write, where appropriate (experimental) • Change the main menu accelerator key
17th Mar 24	1.4.5.1	• Version Control – updated 'diff report' • Compare Module
29th Feb 24	1.4.5.0	• Split lines • Call hierarchy • List all procedure / property / variable / constant Attributes in a Module • Clean Project • Conditional compilation directives … enhancements to expression intrepretation • Backup ... improved • Live Code Info … removed

27th Jan 24	1.4.4.0	• Version Control • Indent for whole Project • Bug fixes including #21, #22
26th Nov 23	1.4.3.0	• Indent procedure / Module • Bug fixes including #17, #18, #19, #20
14th Nov 23	1.4.2.0	• Expanded 'AutoText' • Bug fixes including #16
26th Oct 23	1.4.1.4	• Bug fixes only
23rd Oct 23	1.4.1.3	• Store / restore conditional compilation arguments
4th Oct 23	1.4.1.2	• Orphan-checking of Modules is now all Modules, not just standard Modules • Bug fixes including #13, #14
29th Aug 23	1.4.1.1	• Bug fixes only
7th Aug 23	1.4.1.0	• Sub to 'Run Again' • 'Sub to run' enhancements • Comment / Uncomment • Project Picker
22nd Jun 23	1.4.0.1	• Keyboard shortcuts applied 'statically' are now treated as references (Excel only) • Setting to allow/block automatic checks for updated versions of VBE_Extras
12th Jun 23	1.4.0.0	• Multi-Project … handles references to / from other VBA Projects (including indirect references) • Bug fixes including #11, #12
20th Apr 23	1.3.1.0	• Implement Interface / Extract Interface • DefType statements handled • Extract literal value to Const now allows 'case sensitive' when working with Strings • When showing a list of references, indicate whether the reference is getting or letting/setting (experimental) • Bug fixes including #9, #10
7th Apr 23	1.3.0.0	• Live Code Info • Make 'Option Explicit' (experimental) • New search methodology (experimental) • Major performance improvements when identifying declarations and references
23rd Mar 23	1.2.0.1	• Save / load / reset Settings • Performance improvements when identifying declarations and references
10th Mar 23	1.2.0.0	• Now works in Access • Go to implementation / go to base (for interfaces) • Option to visually distinguish Class Modules that have VB_PredeclaredId = True vs False • Expanded Toolbar • Performance improvements when identifying declarations and references
21st Feb 23	1.1.6.2	• View 'Previous Lists'
15th Feb 23	1.1.6.1	• Improved handling for bang operator ("!") • View references being renamed from the 'Rename' dialog • When viewing references on a line with continuations, see full (continued) line in a tool-tip • More progress bars for potentially long-running operations • Bug fixes including #7, #8
10th Jan 23	1.1.6.0	• Extract literal value to Const • Focus mode • Highlight - draw a box over the text • Show hidden members (auto or via menu)
17th Nov 22	1.1.5.0	• Tables – go to Shape / usages (Excel only) • Names – go to Shape / usages (Excel only) • Tables, Names and Shapes – works across AddIn and active Workbook • List all matches of a literal value
4th Nov 22	1.1.4.1	• Mouse-click navigation / actions • Option to disable Module Picker
25th Oct 22	1.1.4.0	• Installation does not require admin privileges • Themes
20th Sep 22	1.1.3.1	• Bug fixes only

13th Sep 22	1.1.3.0	• Shapes – go to Shape / usages / OnAction (Excel only, single Workbook only) • Orphans - optionally ignore RibbonX callbacks and procedures 'called by String' • Tasks – more priorities • Tidied up menus • The 'Option Explicit' dialog has an option to defer subsequent appearance of itself • Bug fixes including #5
19th Aug 22	1.1.2.1	• Go to declaration above / below • Performance improvements when getting references • Bug fixes including #4
9th Aug 22	1.1.2.0	• Bug fixes only
8th Aug 22	1.1.1.0	• Move lines up / down • Replace All Modules (experimental)
27th Jul 22	1.1.0.0	• Auto add Property • Auto add Factory • Qualify reference • CSV export from orphans dialog • Bug fixes including #3
18th Jul 22	1.0.0.0	• Version 1 release

Appendix 4 – changing the main menu's 'x' accelerator key

The default accelerator key / letter used to trigger the main menu is 'x' (e.g. if you press and release the Tab key and then press and release the 'x' key, the VBE will activate the VBE_Extras main menu). If 'x' does not work for you (some non-English languages have other menu items that already use the 'x' key or you may use other Add-Ins that also uses the 'x' key) then you can change it in the 'App' tab in VBE_Extras' Settings dialog, select the 'Menu accelerator key' button and in the drop-down select the key/letter that you want to use in place of 'x' (or no accelerator key/letter).

Appendix 5 – 'new Outlook'

VBE_Extras does not work with 'new Outlook' (only with what Microsoft are now calling 'classic Outlook') as it does not support COM AddIns (which VBE_Extras is) nor does it support use of VBA code. See https://support.microsoft.com/en-gb/office/getting-started-with-the-new-outlook-for-windows-656bb8d9-5a60-49b2-a98b-ba7822bc7627 for further details.

As of the time of writing this, 'classic Outlook' is used in all 'one time purchase' versions of Office (including 2016, 2019 and 2021) whereas users of Office 365 can choose between 'classic Outlook' and 'new Outlook'.

Appendix 6 – relating to Version Control and Restore

This appendix provides further information relevant to the Version Control and Restore commands.

Relevant to host applications that support UserForms

UserForms are exported as two files (in the same way that they are if you export a UserForm directly using the VBE) … both an .frm file (for the VBA code and Attributes) and an .frx file (for the UI elements).

The VBE itself creates the .frx file (it isn't created by VBE_Extras) and it uses an undocumented binary format meaning that the file cannot be parsed by VBE_Extras; additionally, it includes data that is unique to the saved instance (a timestamp?) and, as such, it always appears to be different every time that an export is created AND will always be different when compared to the existing UserForm in the VBE.

As such, when importing / restoring a UserForm Module (or a Project that includes one or more UserForms) then the Diff analysis will always indicate that the UserForm has been updated (as the data will no longer match). This is particularly problematic when using the Plain type of Version Control combined with Git – as such, when using Version Control, the 'Export UserForm .frx files' setting allows you to either always export .frx files or ask before exporting .frx files.

Document Modules in host applications other than Access

Document Modules cannot actually be imported / restored (the VBE does not allow it as they are controlled from the host application's user interface, not from the VBE). For example, in Excel, Worksheets cannot be added or removed during an import / restore.

As such, an import / restore can only replace the code in the existing Document Module(s) with the code from the exported version. User interface elements cannot be imported / restored, neither can Properties (as seen in the VBE's Properties Window) and neither can Attributes.

As Outlook and Word can only have 1 Document Module then the adding and removing of Document Modules is less of an issue (in Outlook, 'ThisOutlookSession' cannot be renamed; in Word, 'ThisDocument' can be renamed and so it is possible for the Diff analysis to indicate a change of Document Module … but I strongly recommend you do not rename it).

Access-specific

Forms and Report can be exported and imported / restored … including the user interface elements, their Properties and their Attributes. Behind-the-scenes, Access's built-in (but hidden) SaveAsText and LoadFromText functions are used.

Occasionally, in Access, an import / restore can take a number of seconds if Reports are included and if Access decides it needs to get a printer connection (you may see a " Waiting for printer connection ..." dialog). Please be patient as VBE_Extras needs to wait for Access to finish before it can complete the import / restore process.

The Module or Modules being exported must not include characters that are not allowed in Windows filenames (Access can include such characters in Module names). You will not be able to export / created a restore point (and a dialog will be shown explaining the problem) if this is the case.

Case-only Module changes

It is possible for the code in a Module to be changed without any direct editing of the code. For example, if multiple Modules include separate declarations with the same name, changing the case of one will change the case of the other (due to the VBE's case agnostic / case consistency rules). For example, if you add a Sub "Foo" in Module1 and add a Sub "Foo" in Module2 and then change the name in Module1 to "FOO" then the name will also change in Module2.

This means that if you import / restore a previously exported Project, the Diff analysis may indicate that some Modules have changed when you believe they have not … it is the VBE itself that has changed the casing. When this is identified, the Diff analysis will identify the 'changed' Modules as having "case only" changes. When using Version Control you will be given an option to import, or not, the Modules with case-only changes.

Conversely, if you perform a Version Control import but choose not to import Modules with "case only" changes, those changes to the case may still be made (by the VBE itself, not directly by the import) … for example, in a Project with two Modules, 'Module1' and 'Module2':

- If 'Module1' includes a material (i.e. not just a change to the case) change and it includes a declaration for which only the case has changed; and
- If 'Module2' includes a reference to the declaration in 'Module1' for which the case has changed and it includes no other material changes
- … then: if you perform an import that excludes "case only" changes (i.e. it includes 'Module1' but excludes 'Module2') then, once 'Module1' is imported, the VBE will change the case of the reference in 'Module2' to match the case of the declaration in 'Module1' (again, due to the VBE's case agnostic / case consistency rules)

Other info

For all except Access Forms and Reports, the name of the imported / restored Module is controlled by the VB_Name Attribute, not the filename. For Access Forms and Reports, the name is controlled by the filename which must start with "Form_" or "Report_" respectively.

Document Modules are exported with 'custom' file extensions so that their type can be determined before the file contents are read:

- In all applications other than Access, a ".dcm" file extension is used instead of the typical ".cls"
- In Access, a ".afm" extension is used for Forms and a ".art" extension is used for Reports instead of the typical ".txt"

The VBE has a habit of inserting lines at the end of all Modules and at the start of UserForm code Modules when importing / restoring. As such, whitespace before the first line of code and after the last line of code is ignored.

Appendix 7 - the "Microsoft Sans Serif" font

VBE_Extras requires the font "Microsoft Sans Serif" to be installed on your device.

This is a 'protected system font' and so should be pre-installed on all devices. However, it has been confirmed that, for a very small proportion of devices, this is not always the case (see bug number #26 at https://github.com/john-tvh/VBE_Extras/issues).

You can check for the presence of the font in your device's Settings > Fonts then scroll down through the list of fonts (or search) to find "Microsoft Sans Serif" (or "Microsoft Sans Serif Regular") ... note that "MS Sans Serif" and "MS Reference Sans Serif" are not the same font.

If the font is missing, you can download it from the Microsoft website at https://learn.microsoft.com/en-us/typography/font-list/microsoft-sans-serif then click on your version of Windows then scroll down to find "Microsoft Sans Serif" (a file named Micross.ttf): download it and double-click to install it. Re-checking your device's Settings, you should now see the font listed.

Appendix 8 – wildcards in 'filter text boxes'

Filter text boxes are the boxes you can type into at the top of a dialog box and allow you to filter a list of items … items can be declarations, references, Tasks, commands, members, settings … and many more. All filter text boxes match ignoring case.

All filter text boxes allow the use of two wildcards – "?" and "*" … the former works just as you might expect while the latter is slightly different to how Microsoft normally implements wildcard matching (see underlining):

- Question mark ("?") … matches a single character … e.g. "d?g" matches both "dog" and "dig"
- Asterisk ("*") … matches zero, one or more characters with the text on either side of the asterisk being allowed to be in any order e.g. "get*element" matches "getElement", "getAllElements", "elementGet" and "elementsGet"