The Excel / VBA Programming Environment

(C) (P) 2024 Bernd Plumhoff Status: November 17th, 2024

Abstract

With Visual Basic for Applications (VBA), tasks can be automated in the spreadsheet Excel, and special functionalities can be programmed that are not included in Excel's standard features.

Here, I will show programs that I found useful when I had to plan, implement, and execute business processes.

Table of Contents

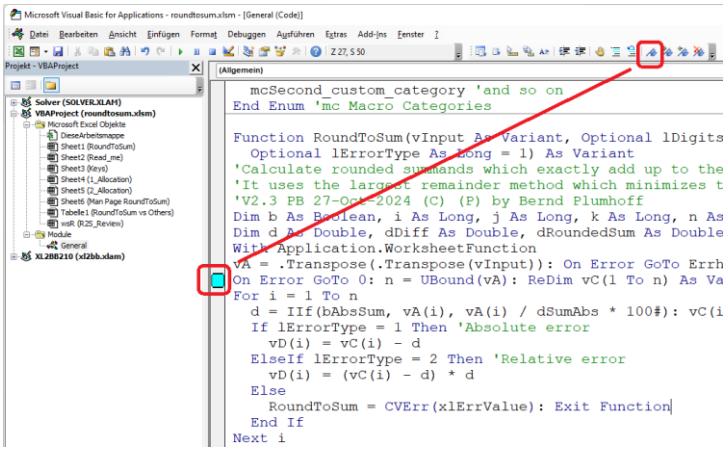
Basics

During Editing

With `Option Explicit` at the beginning of any VBA module you enforce the declataion of all variables used. If you do not use this you should not program.

When the cursor is located at a variable or at an object during editing you can to its declaration with t SHIFT plus the function key F2 (SHIFT + F2). This works for Dim, not ReDim. With STRG + SHIFT + F2 you can jump back to where you were coming from.

In the VBA editor you can mark a code line with the „blue flag“ symbol. With the adjacent flag pointer symbols you can then jump from one to the next or to the previous flag mark:



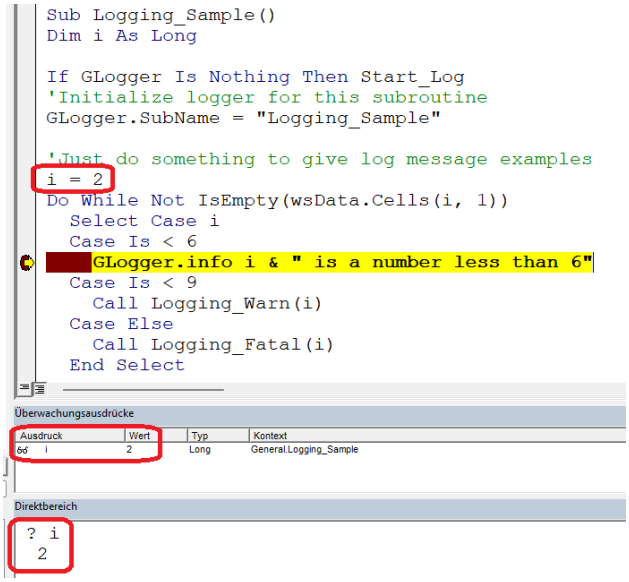
The VBA command `Stop` or a breakpoint set to the left of a code line with stop the execution of a program:

```
Sub Logging_Sample()
Dim i As Long

If GLogger Is Nothing Then Start_Log
'Initialize logger for this subroutine
GLogger.SubName = "Logging_Sample"

'Just do something to give log message examples
i = 2
Do While Not IsEmpty(wsData.Cells(i, 1))
  Select Case i
  Case Is < 6
    Stop
    GLogger.info i & " is a number less than 6"
  Case Is < 9
    Call Logging_Warn(i)
  Case Else
    Call Logging_Fatal(i)
  End Select
  i = i + 1
Loop
```

You can then examine the contents of variables or of Exceol sheets, for example.

During Program Execution

You can interrupt the execution of a VBA program by pressing the ESC key, by the command `Stop` , by a breakpoint or by defining an interrupt condition such as a variable value. When the execution is interrupted the corresponding code line is marked yelllow:



You can now move the cursor to the variable *i* at the point „i = 2“ (do not click, just hover over it). A small window showing i = 2 will appear. You can also select i, right-click, and add a watch for i. Then, the value of i will be displayed in the Watch Expressions window.

With CTRL + g, after a program break, you can go to the Immediate Window. Here, you can enter individual program commands, such as "Print i" (or simply " ? i" — the question mark is shorthand for the Print command).

After a program break, you can continue running the program with the F5 function key. However, you can also step through it one command at a time using F8. If you use SHIFT + F8 instead of F8, the program will not jump into subroutines but will execute them as a single command.

Good Programming Practices

Be a Good Programmer

The most important aspect of a good VBA program is that it is a good program, not just full of VBA tricks. If you don't know associative arrays or classes, you'll be crawling on the floor and never get your application off the ground.

Good Excel and VBA Knowledge

You should have a strong grasp of Excel and VBA. For example, with the VBA command Enum, you can assign names to spreadsheet columns. This makes it easier to modify the program—unless you want to adjust all the hard-coded references to columns when a new column is inserted or an old one is deleted?

Programming Conventions

Learn naming conventions and coding conventions. By following them, you will make your programs more readable, easier to maintain, and more reliable and efficient.

Example links:
https://de.wikibooks.org/wiki/VBA_in_Excel/_Namenskonventionen
https://learn.microsoft.com/de-de/dotnet/visual-basic/programming-guide/program-structure/coding-conventions

Clean Up Macro Recordings

Of course, I will record a macro if I forget a VBA command. But if you use raw, uncleaned spaghetti code from a recorded macro, someone else will have to clean it up for you.

Document Your Program Adequately

Explain what a knowledgeable third party needs to know, but don't write novels about trivialities. Good documentation comes with the program—not after. Only fools and people trying to make themselves indispensable fail to document their work.

Test Your Program Thoroughly

Applications of a certain size require test programs or even a series of regression tests.

Log Your Program Execution

With a Logger class (document the program flow—logging class), you can show an auditor who ran your program, with what parameters, and whether your program ran smoothly.

Optimize Your Program

The quality of your application is largely determined by its design and structure. The more complex the task, the better your expertise and experience should be. By measuring the runtime of individual parts of the program (profiling), you can identify where improvements are still possible.

Jan Karel Pieterse has created a helpful class for profiling:
https://jkp-ads.com/Articles/performanceclass.asp

System Status Save and Restore – SystemState Class

My former colleague Jon T. developed the smallest meaningful VBA class I know: With *SystemState*, you can easily save system status variables and optimize them for your own purposes.

To speed up program execution, you usually write the following at the beginning of a VBA macro:

```
Application.Calculation = xlCalculationManual
Application.ScreenUpdating = False
```

and at the end of the macro:

```
Application.Calculation = xlCalculationAutomatic
Application.ScreenUpdating = True
```

With the *SystemState* class, you only need to write the following at the start:

```
Dim state As SystemState
Set state = New SystemState
'Bitte beachten: Dies kann NICHT mit "Dim state as New SystemState"
abgekürzt werden!
```

and at the end:

```
Set state = Nothing 'Not even necessary - this is done automatically.
```

System Status Variables

The *SystemState* class saves and restores the following system status variables:

Variable	Status	Comment / To Optimize By …
Calculation	xlCalculationAutomatic, xlCalculationManual, xlCalculationSemiautomatic	Determines if recalculation is performed after each cell change. Set to xlCalculationManual.
Cursor	xlDefault, xlBeam, xlNorthwestArrow, xlWait	This is just a display. It's best not to touch it – unless you want to start the debug mode with an hourglass cursor.
DisplayAlerts	Wahr, Falsch	Set to *False* to turn off system prompts.
EnableAnimations	Wahr, Falsch	From Excel 2016 onwards, this disables Excel's screen animations.
EnableEvents	Wahr, Falsch	Set to *False* to prevent event procedures from executing.
Interactive	Wahr, Falsch	Best not to touch – unless you want to prevent all keyboard inputs.
PrintCommunication	Wahr, Falsch	Set to *False* to change page settings without waiting for a response from the printer.
ScreenUpdating	Wahr, Falsch	Set to *False* to turn off screen updates during program execution.
StatusBar	Falsch, "Eine beliebige Benutzerinformation"	The text is displayed in the status bar (bottom of the screen). Set to *False* to clear the display.

SystemState Code

Please copy the following program code into a class module named *SystemState*, not into a regular module.

```
'This class has been developed by my former colleague Jon T.
'I adapted it to newer Excel versions. Any errors are mine for sure.
'Source (EN): http://www.sulprobil.com/systemstate_en/
'Source (DE): http://www.bplumhoff.de/systemstate_de/
'(C) (P) by Jon T., Bernd Plumhoff 3-Nov-2024 PB V1.5
'
'The class is called SystemState.
'It can of course be used in nested subroutines.
'
'This module provides a simple way to save and restore key excel
'system state variables that are commonly changed to speed up VBA code
'during long execution sequences.
'
'Usage:
'    Save() is called automatically on creation and Restore() on destruction
'    To create a new instance:
'        Dim state as SystemState
'        Set state = New SystemState
'    Warning:
'        "Dim state as New SystemState" does NOT create a new instance
'
'
'    Those wanting to do complicated things can use extended API:
'
'    To save state:
'       Call state.Save()
'
'    To restore state and in cleanup code: (can be safely called multiple times)
'       Call state.Restore()
'
'    To restore Excel to its default state (may upset other applications)
'       Call state.SetDefaults()
'       Call state.Restore()
'
'    To clear a saved state (stops it being restored)
'       Call state.Clear()
'
Private Type m_SystemState
  Calculation As XlCalculation
  Cursor As XlMousePointer
  DisplayAlerts As Boolean
  EnableAnimations As Boolean   'From Excel 2016 onwards
  EnableEvents As Boolean
  Interactive As Boolean
  PrintCommunication As Boolean 'From Excel 2010 onwards
  ScreenUpdating As Boolean
  StatusBar As Variant
  m_saved As Boolean
End Type

'
'Instance local copy of m_State?
'
Private m_State As m_SystemState

'
'Reset a saved system state to application defaults
'Warning: restoring a reset state may upset other applications
'
Public Sub SetDefaults()
  m_State.Calculation = xlCalculationAutomatic
  m_State.Cursor = xlDefault
  m_State.DisplayAlerts = True
  m_State.EnableAnimations = True
  m_State.EnableEvents = True
  m_State.Interactive = True
  On Error Resume Next 'In case no printer is installed
  m_State.PrintCommunication = True
  On Error GoTo 0
  m_State.ScreenUpdating = True
  m_State.StatusBar = False
  m_State.m_saved = True 'Effectively we saved a default state
End Sub

'
'Clear a saved system state (stop restore)
'
Public Sub Clear()
    m_State.m_saved = False
End Sub
```

```
'
'Save system state
'
Public Sub Save(Optional SetFavouriteParams As Boolean = False)
  If Not m_State.m_saved Then
    m_State.Calculation = Application.Calculation
    m_State.Cursor = Application.Cursor
    m_State.DisplayAlerts = Application.DisplayAlerts
    m_State.EnableAnimations = Application.EnableAnimations
    m_State.EnableEvents = Application.EnableEvents
    m_State.Interactive = Application.Interactive
    On Error Resume Next 'In case no printer is installed
    m_State.PrintCommunication = Application.PrintCommunication
    On Error GoTo 0
    m_State.ScreenUpdating = Application.ScreenUpdating
    m_State.StatusBar = Application.StatusBar
    m_State.m_saved = True
  End If
  If SetFavouriteParams Then
    Application.Calculation = xlCalculationManual
    Application.DisplayAlerts = False
    Application.EnableAnimations = False
    Application.EnableEvents = False
    On Error Resume Next 'In case no printer is installed
    Application.PrintCommunication = False
    On Error GoTo 0
    Application.ScreenUpdating = False
    Application.StatusBar = False
  End If
End Sub

'
'Restore system state
'
Public Sub Restore()
  If m_State.m_saved Then
    'We check now before setting Calculation because setting
    'Calculation will clear cut/copy buffer
    If Application.Calculation <> m_State.Calculation Then
      Application.Calculation = m_State.Calculation
    End If
    Application.Cursor = m_State.Cursor
    Application.DisplayAlerts = m_State.DisplayAlerts
    Application.EnableAnimations = m_State.EnableAnimations
    Application.EnableEvents = m_State.EnableEvents
    Application.Interactive = m_State.Interactive
    On Error Resume Next 'In case no printer is installed
    Application.PrintCommunication = m_State.PrintCommunication
    On Error GoTo 0
    Application.ScreenUpdating = m_State.ScreenUpdating
    If m_State.StatusBar = "FALSE" Then
      Application.StatusBar = False
    Else
      Application.StatusBar = m_State.StatusBar
    End If
  End If
End Sub

'
'By default save when we are created
'
Private Sub Class_Initialize()
  Call Save(SetFavouriteParams:=True)
End Sub

'
'By default restore when we are destroyed
'
Private Sub Class_Terminate()
  Call Restore
End Sub
```

Documenting Program Flow – *Logging* Class

This Logger class provides logging with the report levels *INFO, WARN, FATAL*, and *EVER*. Program information is logged both in a worksheet and in a file.

The application of this Logger class is not difficult: Simply copy the general module *Logger_Factory* and the class module *Logger* from the example file provided below into your own application, then define the public constant *AppVersion*, for example with the value "My Application Version 1.0" in the main module. After that, you can generate your log messages with the following commands:

```
GLogger.info "Info Meldung ..."
GLogger.warn "Warn Meldung ..."
GLogger.fatal "Fehler Meldung ..."
GLogger.ever "Nichtunterdrückbare Standard Meldung ..."
```

These log messages will be automatically saved in the worksheet Workflow and in the log file “My Application Version 1.0_Logfile_yyyymmdd.log” in the subdirectory *Logs*.

I received the original program code from Cliff G. in 2009 and later extended it. Cliff primarily used this program for debugging. I also find it very useful for logging program executions for revision purposes or to have a program explain its individual execution steps to the user in detail. Additionally, I added version information and system or Excel parameters to quickly identify important differences between different user environments. With this logger, I also typically measure simple runtime durations of SQL queries:

```
'Glogger is declared in module LoggerFactory and set in Sub Start_Log()
Dim dtStamp As Date
'...
dtStamp = Now
'Retrieve data from database here
Glogger.info "SQL xxx ran " & Format(Now - dtStamp, "n:ss") & " [m:ss]"
```

Pros and Cons

In my opinion, this logging program provides the most useful secondary functionality for any VBA application. With it, you can:

- Test in a traceable manner
- Have a program explain all its execution steps in a traceable way
- Easily determine if multiple users are accessing an application simultaneously
- Quickly identify if a user issue is related to a different environment
- Systematically narrow down sporadic application errors
- Convincingly demonstrate to auditors the correct, error-free usage of the program over an extended period (while individual log files can be tampered with, a large number of log files still provides sufficient evidence)
- Roughly determine the runtime of VBA (sub)routines
- Measure the runtime of entire processes

The last point above will raise concerns for works councils and employee representatives:

- When measuring the runtime of entire processes, individual employee performance could be assessed, compared, and potentially used against them.

This would be a clear violation of the GDPR (General Data Protection Regulation), see https://gdpr-info.eu/.

I have never used this logging to measure employee performance or usage, but only for retraining when I identified incorrect usage. However, this should not be taken as an argument for uncritical use.

In my opinion, the approval of works councils or employee representatives can always be obtained by emphasizing the voluntary nature of this self-logging:

- Each user can turn logging on or off before running the program.
- Each user can delete log files at any time afterward.

I have successfully used this logging in multiple countries in Europe (UK, Germany) and with multiple companies (banks, insurance companies, IT providers) without any complaints.

Parameters

Public Constants

AppVersion - This string should contain the program name and its version, e.g.:

```
Public Const AppVersion As String = "... Version x"
```

Then "… Version x" will be logged as version information for this application.

Compiler Constants

Separate_Log_Files_for_each_User - True = Separate daily log files for different users, False = One daily log file for all users

Use_Logger_auto_Open_Close - True will use subs auto_open and auto_close of LoggerFactory, False will not

Logging_on_Screen - Set to True in both LoggerFactory and Logger if you want to log messages also to sheet Workflow (i. e. on screen).

Logging_cashed - Set to True in both LoggerFactory and Logger if you want to speed up the application by writing log messages in one go to the log file at program end. This requires Logging_on_Screen set to True.

Log_WMI_Info - Set to True in LoggerFactory if you like to log interesting Windows Management Instrumentation (WMI) information such as processor, memory, disk, and operating system data.

Show_Reference_Details - True = Show all details, False = Just show description.

Logging Variables

LogFilePath - Full pathname of log file

SubName - Set at the beginning of each subroutine to enable the logger to report on the right subroutine name

LogLevel - The level for which logging should be performed:

1 - Report all log messages: INFO, WARN, FATAL, and EVER
2 - Report all log messages but not INFOs
3 - Report from FATAL level onwards, i. e. just FATAL and EVER messages
4 - Report only EVER messages
5 - Switch off logging

LogScreenRow - Row from where to start logging in sheet Workflow (usually 3).

Sample Output

Standard output for the logs are sheet Workflow the logfile in subfolder Logs:

```
EVER: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - Logging started with Logging_Version_13
EVER: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - Microsoft Windows 11 Home 10.0.22000 (64-Bit)
and Excel 2024 (64-Bit)
INFO: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - Application ThousandsSeparator '.',
DecimalSeparator ',', use system separators
INFO: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - App.Internl ThousandsSeparator '.',
DecimalSeparator ',', ListSeparator ';'
INFO: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - App.Internl xlCountryCode '49',
xlCountrySetting '49'
INFO: BERND-CAPTIVA\earso 12.11.2024 03:57:15 [Start_Log] - VBAProject References: Visual Basic For
Applications, Microsoft Excel 16.0 Object Library, OLE Automation, Microsoft Office 16.0 Object Library
EVER: BERND-CAPTIVA\earso 12.11.2024 03:57:18 [End_Log] - Logging finished with Logging_Version_13
```

Modules

Normal

LoggerFactory contains constants, public variables, default logger settings, and optional autoopen and autoclose subs.

Note: The subroutine Start_Log requires (calls) the subroutines *ApplicationVersion* and *getOperatingSystem* (https://www.devhut.net/vba-determine-the-installed-os/), and the module *LibFileTools* (https://github.com/cristianbuse/VBA-FileTools) to support folder in OneDrive and in Sharepoint.

```
Option Explicit
'This general module is named LoggerFactory. Together with class module Logger it offers logging
functionality.
'Version When         Who             What
'      1 Once upon .. Cliff G.        Initial version
'     13 12-Nov-2024  Bernd Plumhoff  Using LibFileTools https://github.com/cristianbuse/VBA-FileTools,
ApplicationVersion updated
#Const Separate_Logfiles_for_each_User = False
#Const Use_Logger_auto_Open_Close = True 'Enable auto_open and auto_close subs in here
#Const Logging_on_Screen = True          'IMPORTANT: Also change this constant in class module Logger! We
like to see recent run's loggging messages on screen in tab Workflow
#Const Logging_cashed = False            'IMPORTANT: Also change this constant in class module Logger!
Write logging messages into file at program end to speed this up
#Const Log_WMI_Info = False              'True shows interesting Windows Management Instrumentation (WMI)
data
#Const Show_Reference_Details = False    'True: Show all details; False: Just show description
Public GLogger As Logger                 'Global logfile object - variable scope is across all modules
Public GsThisLogFilePath As String
' Constant log levels
Public Const INFO_LEVEL As Integer = 1
Public Const WARN_LEVEL As Integer = 2
Public Const FATAL_LEVEL As Integer = 3
Public Const EVER_LEVEL As Integer = 4 'For logging messages which cannot be switched off
Public Const DISABLE_LOGGING As Integer = 5
'The application-specific defaults
Const DEFAULT_LOG_FILE_PATH As String = "" 'Force error if not set [Bernd 12-Aug-2009]
Const DEFAULT_LOG_LEVEL As Integer = INFO_LEVEL

Public Function getLogger(sSubName As String) As Logger
  Dim oLogger As New Logger
  oLogger.SubName = sSubName
  'Defaults to the specified values - but may be overridden before used
  oLogger.LogLevel = DEFAULT_LOG_LEVEL
  oLogger.LogFilePath = DEFAULT_LOG_FILE_PATH
  Set getLogger = oLogger
End Function

#If Use_Logger_auto_Open_Close Then
  Sub auto_open()
  'Version Date        Programmer Change
  '9       12-Sep-2021 Bernd      Code outsorced to Start_Log so that user does not need to use auto_open
  Start_Log
  End Sub

  Sub auto_close()
  'Version Date        Programmer Change
  '3       12-Sep-2021 Bernd      Code outsorced to End_Log so that user does not need to use auto_close
  End_Log
  End Sub
#End If '#If Use_Logger_auto_Open_Close

Sub Start_Log()
'Version Date        Programmer Change
'3       02-Nov-2023 Bernd      Log interesting Windows Management Instrumentation (WMI) infos
'4       27-Feb-2024 Bernd      Show_Reference_Details added
'5       12-Nov-2024 Bernd      Using LibFileTools https://github.com/cristianbuse/VBA-FileTools
Dim i As Long
Dim s As String, sDel As String, sPath As String
#If Log_WMI_Info = True Then
  Dim oWMISrvEx As Object 'SWbemServicesEx
  Dim oWMIObjSet As Object 'SWbemServicesObjectSet
  Dim oWMIObjEx As Object 'SWbemObjectEx
  Dim oWMIProp As Object 'SWbemProperty
  Dim sWQL As String 'WQL Statement
  Dim v As Variant
#End If
'Need to import for next 3 lines: LibFileTools https://github.com/cristianbuse/VBA-FileTools
sPath = GetLocalPath(ThisWorkbook.Path) & "\Logs"
If Not IsFolder(sPath) Then
  CreateFolder (sPath)
End If
If GLogger Is Nothing Then Set GLogger = New Logger
#If Separate_Logfiles_for_each_User Then
  'If AppVersion is not defined please define it in your main module like:
  'Public Const AppVersion As String = "Application Version ..."
  GLogger.LogFilePath = sPath & "\" & Environ("Userdomain") & _
    "_" & Environ("Username") & "_" & AppVersion & "_" & "Logfile_" & _
    Format(Now, "YYYYMMDD") & ".txt"
#Else
  GLogger.LogFilePath = sPath & "\" & AppVersion & "_" & _
    "Logfile_" & Format(Now, "YYYYMMDD") & ".txt"
#End If
GLogger.LogLevel = 1
#If Logging_on_Screen Then
  GLogger.LogScreenRow = 3
  wsW.Range("E2:E4").ClearContents
  wsW.Range("5:65535").Delete
#End If
'Initialize logger for this subroutine
With Application
```

```
GLogger.SubName = "Start_Log"
GLogger.ever "Logging started with " & AppVersion
#If Log_WMI_Info = True Then
  Set oWMISrvEx = GetObject("winmgmts:root/CIMV2")
  For Each v In Array("BaseService", "Processor", "PhysicalMemoryArray", "LogicalDisk", "OperatingSystem")
    'Not: "NetworkAdapterConfiguration", "VideoController", "OnBoardDevice", "Printer", "Product"
    Set oWMIObjSet = oWMISrvEx.ExecQuery("Select * From Win32_" & v)
    For Each oWMIObjEx In oWMIObjSet
      s = v & ": "
      For Each oWMIProp In oWMIObjEx.Properties_
        If Not IsNull(oWMIProp.Value) Then
          If Not IsArray(oWMIProp.Value) Then
            Select Case v
            Case "BaseService"
              If InStr("'SystemName'", "'" & oWMIProp.Name & "'") > 0 Then
                GLogger.ever oWMIProp.Name & "='" & Trim(oWMIProp.Value) & "'"
                GoTo Next_v
              End If
            Case "Processor"
              If
InStr("'Name'Description'NumberOfEnabledCore'AddressWidth'DataWidth'CurrentClockSpeed'LoadPercentage'", _
                "'" & oWMIProp.Name & "'") > 0 Then
                If IsNumeric(oWMIProp.Value) Then
                  s = s & oWMIProp.Name & "=" & Format(oWMIProp.Value, "#,##0") & ", "
                Else
                  s = s & oWMIProp.Name & "='" & Trim(oWMIProp.Value) & "', "
                End If
              End If
            Case "PhysicalMemoryArray"
              If InStr("'MaxCapacityEx'", _
                "'" & oWMIProp.Name & "'") > 0 Then s = s & oWMIProp.Name & "=" & Format(oWMIProp.Value,
"#,##0") & ", "
            Case "LogicalDisk"
              If InStr("'DeviceID'ProviderName'Size'FreeSpace'", _
                "'" & oWMIProp.Name & "'") > 0 Then
                If IsNumeric(oWMIProp.Value) Then
                  s = s & oWMIProp.Name & "=" & Format(oWMIProp.Value, "#,##0") & ", "
                Else
                  s = s & oWMIProp.Name & "='" & Trim(oWMIProp.Value) & "', "
                End If
              End If
            Case "OperatingSystem"
              If
InStr("'FreePhysicalMemory'FreeVirtualMemory'FreeSpaceInPagingFiles'MaxProcessMemorySize'InstallDate'", _
                "'" & oWMIProp.Name & "'") > 0 Then s = s & oWMIProp.Name & "=" & Format(oWMIProp.Value,
"#,##0") & ", "
            End Select
          End If
        End If
      Next oWMIProp
      If Len(s) > Len(v & ": ") Then GLogger.ever Left(s, Len(s) - 2)
    Next oWMIObjEx
Next_v:
  Next v
#End If
#If Win64 Then
  s = "64"
#Else
  s = "32"
#End If
GLogger.ever getOperatingSystem() & " and " & ApplicationVersion() & _
  " (" & s & "-Bit)" '& .Version & .Build & " (" & .CalculationVersion & ")"
GLogger.info "Application ThousandsSeparator '" & .ThousandsSeparator & _
  "', DecimalSeparator '" & .DecimalSeparator & "', " & _
  IIf(Not (Application.UseSystemSeparators), "do not ", "") & "use system separators"
GLogger.info "App.Internl ThousandsSeparator '" & .International(xlThousandsSeparator) & _
  "', DecimalSeparator '" & .International(xlDecimalSeparator) & "', ListSeparator '" & _
  .International(xlListSeparator) & "'"
GLogger.info "App.Internl xlCountryCode '" & .International(xlCountryCode) & _
  "', xlCountrySetting '" & .International(xlCountrySetting) & "'"
End With
With ThisWorkbook.VBProject.References 'In case of error tick box Trust access to the VBA project object
  'model under File / Options / Trust Center / Trust Center Settings / Macro Settings
  s = "VBAProject References: "
  On Error Resume Next
  For i = 1 To .Count
    #If Show_Reference_Details Then
      GLogger.info s
      s = ""
      s = s & .Item(i).Description
      s = s & ", FullPath: '" & .Item(i).fullPath & "'"
      s = s & ", Guid: " & .Item(i).GUID
      s = s & ", BuiltIn: " & .Item(i).BuiltIn
      s = s & ", IsBroken: " & .Item(i).IsBroken
      s = s & ", Major: " & .Item(i).Major
      s = s & ", Minor: " & .Item(i).Minor
    #Else
      s = s & sDel & .Item(i).Description
      sDel = ", "
    #End If
```

```
  Next i
  GLogger.info s
End With
'Now two examples of environment variables which might not exist for all Windows / Excel installations.
'Use Sub List_Environ_Variables below to see which variables exist on your system.
s = ""
s = Environ("CRC_VDI-TYPE") 'If this does not exist we will not log anything
If s <> "" Then GLogger.info "CRC_VDI-TYPE: '" & s & "'"
s = ""
s = Environ("ORACLE_HOME_X64") 'If this does not exist we will not log anything
If s <> "" Then GLogger.info "Oracle Client: '" & s & "'"
On Error GoTo 0
End Sub

Sub End_Log()
'Change History:
'Version Date        Programmer Change
'1       12-Sep-2021 Bernd      Initial version so that user does not need to use auto_close. He can
manually call this sub.
If GLogger Is Nothing Then Call auto_open
GLogger.SubName = "End_Log"
'If AppVersion is not defined please define it in your main module like: Public Const AppVersion As String
= "Application Version ..."
GLogger.ever "Logging finished with " & AppVersion
#If Logging_cashed Then
  Set GLogger = Nothing 'Necessary, or Class_Terminate() won't be called for GLogger because it's Public
#End If
End Sub
```

A sample module *General* which show how to use the logger:

```
Option Explicit
'Version When         Who             What
'     11 03-Nov-2023  Bernd Plumhoff  Log interesting Windows Management Instrumentation (WMI) infos
'     12 17-Feb-2024  Bernd Plumhoff  Show_Reference_Details added
'     13 12-Nov-2024  Bernd Plumhoff  Using LibFileTools https://github.com/cristianbuse/VBA-FileTools

Public Const AppVersion As String = "Logging_Version_13"

Sub Logging_Sample()
Dim i As Long

If GLogger Is Nothing Then Start_Log
'Initialize logger for this subroutine
GLogger.SubName = "Logging_Sample"

'Just do something to give log message examples
i = 2
Do While Not IsEmpty(wsData.Cells(i, 1))
    Select Case i
    Case Is < 6
        GLogger.info i & " is a number less than 6"
    Case Is < 9
        Call Logging_Warn(i)
    Case Else
        Call Logging_Fatal(i)
    End Select
    i = i + 1
Loop

#If Logging_cashed Then
Set GLogger = Nothing 'Necessary, or Class_Terminate() won't be called for GLogger since it's Public
#End If

End Sub

'You do not need extra subroutines to log warn messages or fatal messages.
'They are just examples of additional subroutines which do some logging.
Sub Logging_Warn(i As Long)
    'Initialize logger for this subroutine
    GLogger.SubName = "Logging_Warn"
    GLogger.warn i & " is 6, 7, or 8"
End Sub

Sub Logging_Fatal(i As Long)
    'Initialize logger for this subroutine
    GLogger.SubName = "Logging_Fatal"
    GLogger.fatal i & " is greater 8"
End Sub
```

Class Modules

Logger contains the logging functionality:

```
Option Explicit
'This class module is named Logger. Together with class module LoggerFactory it offers logging
functionality.
'Version When         Who             What
'      1 Once upon .. Cliff G.        Initial version
'     13 12-NOv-2024  Bernd Plumhoff  Same version as LoggerFactory
#Const Logging_on_Screen = True 'IMPORTANT: Also change this constant in module LoggerFactory! We like to
see recent run's loggging messages on screen in tab Workflow
#Const Logging_cashed = False   'IMPORTANT: Also change this constant in module LoggerFactory! Write
logging messages into file at program end to speed this up
Const INFO_LEVEL_TEXT As String = "INFO:"
Const WARN_LEVEL_TEXT As String = "#WARN:"
Const FATAL_LEVEL_TEXT As String = "##FATAL:"
Const EVER_LEVEL_TEXT As String = "EVER:"
Private sThisSubName As String
Private iThisLogLevel As Integer
#If Logging_on_Screen Then
  Private iThisLogRow As Integer
  Public Property Let LogScreenRow(iLogRow As Integer)
    iThisLogRow = iLogRow
  End Property

  Public Property Get LogScreenRow() As Integer
    LogScreenRow = iThisLogRow
  End Property
#End If

Public Property Let LogFilePath(sLogFilePath As String)
  GsThisLogFilePath = sLogFilePath
End Property

Public Property Get LogFilePath() As String
  LogFilePath = GsThisLogFilePath
End Property

Public Property Let SubName(sSubName As String)
  sThisSubName = sSubName
End Property

Public Property Get SubName() As String
  SubName = sThisSubName
End Property

Public Property Let LogLevel(iLogLevel As Integer)
  iThisLogLevel = iLogLevel
End Property

Public Property Get LogLevel() As Integer
  LogLevel = iThisLogLevel
End Property

Public Sub info(sLogText As String)
  If Me.LogLevel = LoggerFactory.INFO_LEVEL Then
    Call WriteLog(LoggerFactory.INFO_LEVEL, sLogText)
  End If
End Sub

Public Sub warn(sLogText As String)
  If Me.LogLevel < LoggerFactory.FATAL_LEVEL Then
    Call WriteLog(LoggerFactory.WARN_LEVEL, sLogText)
  End If
End Sub

Public Sub fatal(sLogText As String)
  If Me.LogLevel <= LoggerFactory.FATAL_LEVEL Then
    Call WriteLog(LoggerFactory.FATAL_LEVEL, sLogText)
  End If
End Sub

Public Sub ever(sLogText As String)
  If Me.LogLevel <= LoggerFactory.EVER_LEVEL Then
    Call WriteLog(LoggerFactory.EVER_LEVEL, sLogText)
  End If
End Sub
```

```
Private Sub WriteLog(iLogLevel As Integer, sLogText As String)
  Dim FileNum As Integer, LogMessage As String, sDateTime As String, sLogLevel As String
  Select Case iLogLevel
  Case LoggerFactory.INFO_LEVEL
    sLogLevel = INFO_LEVEL_TEXT
  Case LoggerFactory.WARN_LEVEL
    sLogLevel = WARN_LEVEL_TEXT
  Case LoggerFactory.FATAL_LEVEL
    sLogLevel = FATAL_LEVEL_TEXT
  Case LoggerFactory.EVER_LEVEL
    sLogLevel = EVER_LEVEL_TEXT
  Case Else
    sLogLevel = "!INVALID LOG LEVEL!"
  End Select
  sDateTime = CStr(Now())
  LogMessage = sLogLevel & " " & Environ("Userdomain") & "\" & Environ("Username") & " " & _
    sDateTime & " [" & Me.SubName & "] - " & sLogText
  #If Not Logging_cashed Then
    FileNum = FreeFile
    Open Me.LogFilePath For Append As #FileNum
    Print #FileNum, LogMessage
    Close #FileNum
  #End If
  #If Logging_on_Screen Then
    wsW.Cells(iThisLogRow, 5) = LogMessage
    iThisLogRow = iThisLogRow + 1
  #End If
End Sub

Private Sub Class_Initialize()
  #If Logging_cashed And Not Logging_on_Screen Then
    Err.Raise Number:=vbObjectError + 513, Description:="Logging_cashed requires Logging_on_Screen"
  #End If
End Sub

Private Sub Class_Terminate()
  #If Logging_cashed Then
    Dim i As Long, FileNum As Integer, LogMessage As String
    FileNum = FreeFile
    Open Me.LogFilePath For Append As #FileNum
    For i = 3 To iThisLogRow - 1
      LogMessage = wsW.Cells(i, 5).Text
      Print #FileNum, LogMessage
    Next i
    Close #FileNum
  #End If
End Sub
```