Die Excel / VBA Programmierumgebung

(C) (P) 2024 Bernd Plumhoff Stand: 17. November 2024

Abstract

Mit Visual Basic for Applications (VBA kann man in der Tabellenkalkulation Excel Aufgaben automatisieren und spezielle Funktionalitäten programmieren, die im Funktionsumfang von Excel nicht enthalten sind.

Hier zeige ich Programme, die ich nützlich fand, als ich Betriebsprozesse planen, umsetzen und ausführen musste.

Inhaltsverzeichnis

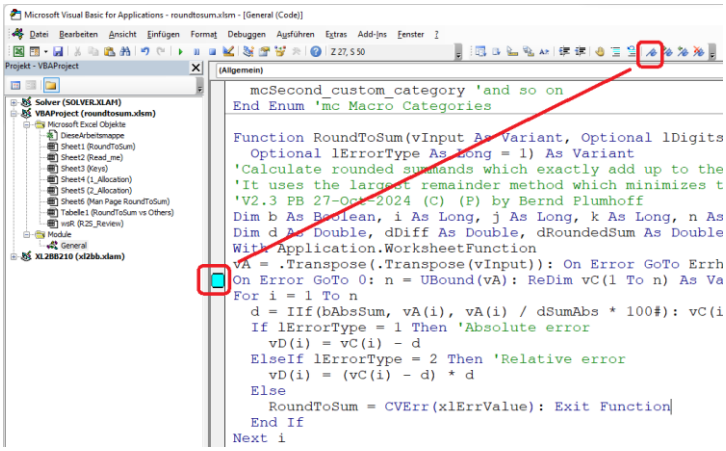
Grundlegendes

Während des Editierens

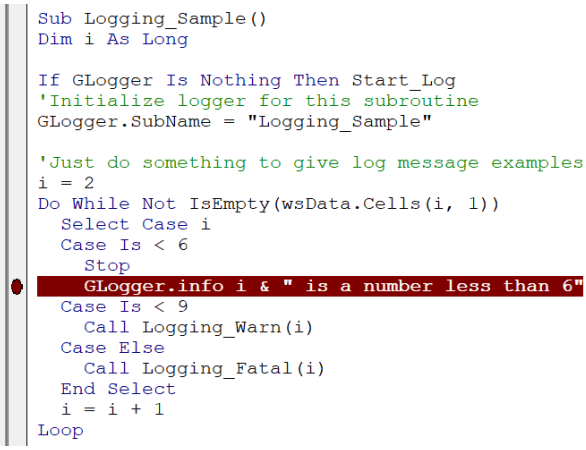
Mit `Option Explicit` erzwingen Sie in einem VBA Modul die Deklaration von allen verwendeten Variablen. Wer dies nicht verwendet, sollte nicht programmieren.

Wenn der Cursor in der VBA Programmierungebung auf einer Variablen oder einem Objeckt steht, springen Sie mit SHIFT und der Funktionstaste F2 (SHIFT + F2) direkt zur Deklaration dieser Variablen (nur Dim, nicht ReDim). Mit STRG + SHIFT + F2 springen Sie zurück.

Im VBA Editor können Sie mit dem Symbol „blaue Flagge“ Textstellen markieren, zu denen Sie mit den angrenzenden Zeigersymbolen schnell springen können:



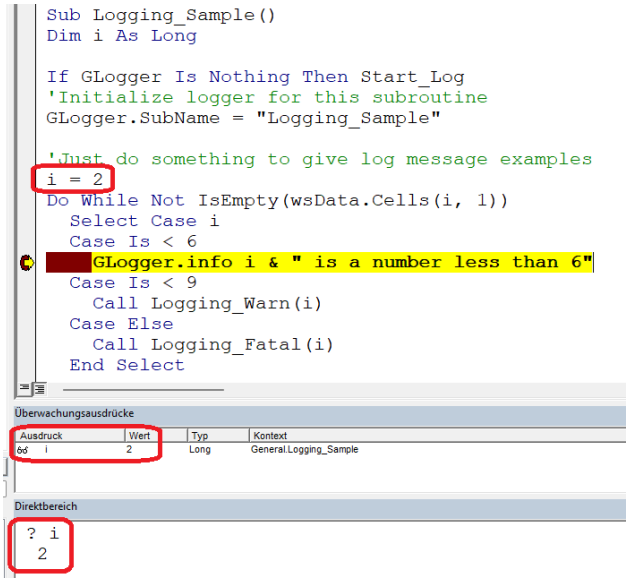
Mit dem Programmbefehl `Stop` oder wenn Sie auf den grauen Bereich gleich links neben einer Programmzeile klicken, können Sie einen Stopppunkt (engl. breakpoint) setzen:



Wenn das Programm später ausgeführt wird, dann stoppt es an dieser Stelle, und Sie können z. B. den Inhalt von Variablen oder Tabellenblättern prüfen.

Während der Programmausführung

Wenn ein VBA Programm ausgeführt wird, können Sie es durch Drücken der ESC Taste unterbrechen. Auch der Befehl `Stop` oder ein Stopppunkt unterbricht das Programm, wenn es an die entsprechende Stelle gelangt, und kennzeichnet die aktuelle Programmzeile gelb:



Sie können nun mit dem Cursor zur Variablen i an der Stelle „i = 2“ gehen (nicht klicken, lediglich darüber schweben, „hovern“). Es wird ein kleines Fenster i = 2 gezeigt. Sie können auch i auswählen, die rechte Maustaste drücken und eine Überwachung für i hinzufügen. Dann wird der Wert von i im Fenster Überwachungsausdrücke gezeigt.

Mit STRG + g kommen Sie nach einer Programmunterbrechung in den Direktbereich (engl. Immediate Window). Hier können Sie einzelne Programmbefehle eingeben wie z. B. „Print i“ (oder einfach kurz „? I“ – das Fragezeichen ist das Kürzel für den Befehl Print).

Sie können nach einer Programmunterbrechung das Programm mit der Funktionstaste F5 weiterlaufen lassen. Sie können es aber mit F8 auch weiter schrittweise Befehl-für-Befehl ausführen. Mit SHIFT + F8 an Stelle von F8 verzweigt das Programm nicht in Unterprogramme, sondern führt diese als einen Befehl aus.

Gute Programmierpraxis

Seien Sie ein guter Programmierer

Das Wichtigste an einem guten VBA Programm ist, dass es ein gutes Programm ist, und nicht voller VBA Tricks. Wenn Sie keine assoziativen Arrays und keine Klassen kennen, dann müssen sie auf dem Boden herumkriechen und bekommen Ihre Anwendung nie zum Fliegen.

Gutes Excel und VBA Wissen

Sie sollten Excel and VBA gut beherrschen. Mit dem VBA Befehl Enum können Sie z. B. Tabellen-spalten Namen geben. Dies vereinfacht Programmänderungen - oder wollen Sie alle hart kodierten Verweise auf Spalten rechts von einer neu eingefügten oder gelöschten Spalte anpassen?

Programmierkonventionen

Lernen Sie Namenskonventionen und Kodierkonventionen. Durch ihre Anwendung machen Sie Ihre Programme lesbarer, einfacher zu warten, und verlässlicher und effizienter.
Beispiellinks:
https://de.wikibooks.org/wiki/VBA_in_Excel/_Namenskonventionen
https://learn.microsoft.com/de-de/dotnet/visual-basic/programming-guide/program-structure/coding-conventions

Säubern Sie Makroaufzeichnungen

Selbstverständlich nehme ich ein Makro auf, wenn ich einen VBA Befehl vergessen habe. Aber wenn Sie aufgezeichneten ungesäuberten Spaghetticode verwenden, muss dies Ihr Nachfolger aufräumen.

Dokumentieren Sie Ihr Programm ausreichend

Erklären Sie was ein kundiger Dritter wissen muss, aber schreiben Sie keine Romane über Trivialitäten. Gute Dokumentation kommt mit dem Programm - nicht danach. Nur Dummköpfe und Menschen, die sich unentbehrlich machen wollen, dokumentieren nicht.

Testen Sie Ihr Programm gut

Anwendungen ab einer gewissen Größe benötigen Testprogramme oder sogar eine Serie von Regressionstests.

Protokollieren Sie Ihre Programmausführung

Mit einer Logger Klasse (Programmablauf dokumentieren – *Logging* Klasse) können Sie einem Revisor zeigen, wer Ihr Programm mit welchen Parametern ausführte und ob Ihr Programm problemlos durchlief.

Optimieren Sie Ihr Programm

Die Qualität Ihrer Anwendung wird maßgeblich durch ihr Design und ihre Struktur bestimmt. Je komplexer die Aufgabe, desto besser sollte Ihr Fachwissen und Ihre Erfahrung sein. Durch das Messen der Laufzeiten einzelner Programmteile (engl. Profiling) können Sie erkennen, wo noch Verbesserungen möglich sind.

Jan Karel Pieterse hat eine hilfreiche Klasse für das Profiling erstellt:
https://jkp-ads.com/Articles/performanceclass.asp

Systemstatus sichern und zurückschreiben – *SystemState* Klasse

Mein ehemaliger Kollege Jon T. entwickelte die kleinste mir bekannte sinnvolle VBA Klasse: Mit *SystemState* kann man Systemstatusvariablen leicht speichern und für eigene Zwecke optimieren.

Um die Programmausführung zu beschleunigen, schreibt man normalerweise zu Beginn eines VBA Makros

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

und am Ende des Makros

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

Mit dem Klassenmodul *SystemState* schreibt man am Start lediglich

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

und am Ende

```
Set state = Nothing 'Nicht einmal nötig - dies wird automatisch gemacht
```

Systemstatus Variablen

Die Klasse *SystemState* sichert und restauriert die folgenden Systemstatus Variablen:

Variable	Status	Kommentar / Zu optimieren durch …
Calculation	xlCalculationAutomatic, xlCalculationManual, xlCalculationSemiautomatic	Bestimmt ob nach jeder Zelländerung eine Neuberechnung durchgeführt wird. Auf xlCalculationManual setzen.
Cursor	xlDefault, xlBeam, xlNorthwestArrow, xlWait	Dies ist lediglich eine Anzeige. Am besten nicht anfassen - es sei denn, man möchte den Debug Modus z. B. mit einem Sanduhr Zeiger beginnen.
DisplayAlerts	Wahr, Falsch	Auf Falsch (engl. False)setzen um Systemrückfragen abzuschalten.
EnableAnimations	Wahr, Falsch	Ab Excel Version 2016 kann man hiermit Excel's Bildschirmanimationen abschalten.
EnableEvents	Wahr, Falsch	Auf Falsch (engl. False)setzen um Ereignisprozeduren an der Ausführung zu hindern.
Interactive	Wahr, Falsch	Am besten nicht anfassen - es sei denn, man möchte unbedingt alle Tastatureingaben verhindern.
PrintCommunication	Wahr, Falsch	Auf Falsch (engl. False)setzen um Seiteneinstellungen zu ändern ohne auf Rückantwort vom Drucker zu warten.
ScreenUpdating	Wahr, Falsch	Auf Falsch (engl. False) setzen um Bildschirmaktualisierungen während der Programmausführung abzuschalten.
StatusBar	Falsch, "Eine beliebige Benutzerinformation"	Der Text wird in der Statuszeile (unterste Bildschirmzeile) gezeigt. Auf Falsch (engl. False) setzen um die Anzeige zu löschen.

SystemState Programmcode

Bitte kopieren Sie den folgenden Programmcode in ein Klassenmodul mit dem Namen *SystemState*, nicht in ein normales Modul.

```
'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
```

Programmablauf dokumentieren – *Logging* Klasse

Diese Logger Klasse bietet Logging mit den Berichtsstufen INFO, WARN, FATAL und EVER an. Die Programminformationen werden sowohl in einem Tabellenblatt als auch in einer Datei festgehalten.

Die Anwendung dieser Logger Klasse ist nicht schwer: Einfach das allgemeine Modul Logger_Factory und das Klassenmodul Logger aus der unten bereitgestellten Beispieldatei in die eigene Anwendung kopieren, dann die Public Const AppVersion zum Beispiel mit dem Wert “Meine Anwendung Version 1.0” im Hauptmodul definieren, und dann kann man mit

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

die eigenen Logmeldungen erzeugen und automatisch im Tabellenblatt Workflow und in der Logdatei “Meine Anwendung Version 1.0_Logfile_yyyymmdd.log” im Unterverzeichnis Logs speichern.

Ich erhielt den ursprünglichen Programmcode 2009 von Cliff G. und erweiterte ihn später. Cliff verwendete dieses Programm hauptsächlich zum Debuggen. Ich finde es auch sehr sinnvoll, um Programmläufe zu Revisionszwecken zu protokollieren, oder damit ein Programm dem Benutzer ggf. detailliert seine einzelnen Ausführungsschritte erläutert. Weiter fügte ich Versionsinformationen und System- oder Excel-Parameter hinzu, um rasch wichtige Unterschiede zwischen verschiedenen Benutzerumgebungen zu ermitteln. Mit diesem Logger messe ich gewöhnlich auch einfache Laufzeiten von SQL Abfragen:

```
'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]"
```

Für und Wider

Dieses Logging Programm bietet meines Erachtens die sinnvollste sekundäre Funktionalität für jede VBA Anwendung. Man kann:

- nachvollziehbar testen
- ein Programm alle seine Ausführungsschritte nachvollziehbar erklären lassen
- einfach feststellen, ob mehrere Anwender eine Anwendung gleichzeitig nutzen
- leicht erkennen, ob ein Anwenderproblem auf einer unterschiedlichen Umgebung beruht
- auch sporadische Anwendungsfehler systematisch eingrenzen
- über einen längeren Zeitraum hinweg auch Revisoren überzeugend die korrekte fehlerfreie Nutzung nachweisen (einzelne Logdateien können zwar manipuliert werden, aber eine größe Menge von Logdateien wirkt dennoch hinreichend überzeugend)
- die Laufzeit von VBA (Unter-)Routinen grob bestimmen
- die Durchlaufzeiten von gesamten Prozessen messen

Der letzte obige Punkt wird Betriebs- und Personalräte hellhörig machen:

- wenn man Durchlaufzeiten von ganzen Prozessen misst, könnte man die Leistung einzelner Mitarbeiter ermitteln, vergleichen und ggf. gegen sie verwenden.

Dies wäre ein klarer Verstoß gegen die DSGVO (Datenschutz-Grundverordnung), siehe https://dsgvo-gesetz.de/

Ich habe dieses Logging nie gegen meine Mitarbeiter oder Anwender für eine Leistungsmessung verwendet, sondern lediglich für Nachschulungen genutzt, wenn ich fehlerhafte Nutzungen erkannte. Aber dies kann selbstverständlich nicht als Argument für eine unbedenkliche Nutzung dienen.

Eine Zustimmung von Betriebs- und Personalräten kann m. E. immer erreicht werden, wenn man auf die Freiwilligkeit dieser Selbstaufschreibung hinweist:

- jeder Anwender kann das Logging vor einem Programmlauf ein- oder ausschalten
- jeder Anwender kann die Log-Dateien zu jeder Zeit im Nachhinein löschen

Ich setzte und setze in Europa in mehreren Ländern (UK, Deutschland) bei mehreren Gesellschaften (Banken, Versicherungen, IT Providern) dieses Logging ohne jede Beanstandung erfolgreich ein.

Parameter

Public (öffentliche) Konstanten

AppVersion - Diese Zeichenkette sollte den Programmnamen und seine Version enthalten, z. B.:

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

Dann wird "… Version x" als Versionsinformation für dieses Programm protokolliert.

Compilerkonstanten

Separate_Log_Files_for_each_User - True erstellt für jeden Benutzer eigene Logdateien, False führt zu einer täglichen Logdatei für alle Benutzer

Use_Logger_auto_Open_Close - True verwendet die Subroutinen auto_open und auto_close im Modul LoggerFactory, False nicht.

Logging_on_Screen - In LoggerFactory und Logger auf True setzen um Nachrichten auch im Tabellenblatt Workflow zu zeigen.

Logging_cashed - In LoggerFactory und Logger auf True setzen um das Logging zu beschleunigen. Log Nachrichten werden dann erst am Ende des Programlaufs in eine Datei geschrieben. Hierfür muss auch Logging_on_Screen auf True gesetzt werden.

Log_WMI_Info - In LoggerFactory auf True setzen um interessante Windows Management Instrumentation (WMI) Informationen auszugeben wie z. B. Prozessor-, Speicher-, Laufwerks- und Betriebssystem-Angaben.

Show_Reference_Details - True zeigt alle Details, False zeigt lediglich die Beschreibung.

Logging Variablen

LogFilePath - Vollständiger Pfadname der Logdatei

SubName - Muss am Anfang jeder Subroutine gesetzt werden um den Sub Namen korrekt im Log zu protokollieren

LogLevel - Die Logging Berichtsstufen:

```
1 - Alle Log Nachrichten protokollieren: INFO, WARN, FATAL, and EVER
2 - Alle Log Nachrichten mit Ausnahme von Stufe INFO protokollieren
3 - Nur FATAL und EVER Log Nachrichten protokollieren
4 - Lediglich EVER Log Nachrichten protokollieren
5 - Kein Logging
```

LogScreenRow - Startzeile für das Logging in Tabellenblatt Workflow (gewöhnlich 3)

Beispielausgabe

Die Logs werden standardmäßig im Tabellenblatt Workflow und in der Logdatei im Unterverzeichnis Logs ausgegeben:

```
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
```

Module

Die Logs werden standardmäßig im Tabellenblatt Workflow und in der Logdatei im Unterverzeichnis Logs ausgegeben:

Normal

LoggerFactory enthält Konstanten, öffentliche Variablen, Standard Logger Einstellungen und optionale Auto-Open and Auto-Close Subroutinen.

Hinweis: Die Prozedur *Start_Log* benötigt (ruft auf) die Prozeduren *ApplicationVersion* und *getOperatingSystem* (https://www.devhut.net/vba-determine-the-installed-os/) sowie das Modul *LibFileTools* (https://github.com/cristianbuse/VBA-FileTools), um auch Verzeichnisse unter OneDrive und Sharepoint zu unterstützen.

```
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
```

Ein Beispielmodul General welches zeigt wie man den Logger nutzen kann:

```
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
```

Klassenmodule

Logger enthält die Logging Funktionalität:

```
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
```