Logging for VBA

Contents

Introduction

This VBA Logging framework is especially designed for VBA Applications. It provides feature specific to VBA development that include:

- direct replacement of 'Debug.Print txt' with 'Logging.log (txt)'
- direct Logging with 'logpoint information'
- logging to a Logbuffer - with the option to create a Trace file of the Log
- full Log4VBA style logging.

Different to logging frameworks like log4J it is not necessary to instantiate a 'Logger' Object prior to log a message: A simple procedure call is sufficient. If preferred the VBA Logging Framework also provides the sophisticated approach creating a Logger Object instance with the class name (comparable to log4J). Both approaches can be combined. Please refer to the Examples and the Unit Tests for more details.

Download source files:

- Logging.bas
- Logger.cls
- Logbuffer.cls
- Unit Tests (Optional)

Download as XLA library:

- Logging.xla

Examples:

Replacing Debug.Print

After installing the Logging Module "Debug.Print" statements can be replaced as:

```
instead
  Debug.Print txt
use
  Logging.log (txt)
```

Note: It is not necessary to initialize a Logger! The Resulting 'printout' will be the same as Debug.Print but with the option to log to a file or the Logbuffer.

A more usefull Logging statement is a call like

```
Logging.logINFO ("myinfotxt..")
```

or adding Logpoint Information

```
Logging.logINFO "This is my message ..", "MySubOrFunction"
```

wich will give a result like

```
(28.08.2008 10:53:20) INFO:  myinfotxt..
(28.08.2008 10:53:20)[MySubOrFunction]-INFO:  This is my message ..
```

These messiges will be logged only if 'LOG_LEVEL = INFO' or finer.

Log4VBA style logging:

```
Dim myLogger As Object 'globally define

' initialize Logger and set Module Name for example 'VBALogger'
Set myLogger = Logging.getNewLogger(Application.VBE.ActiveVBProject.name)

' log  ALL to Console, Buffer, File
Call myLogger.setLoggigParams(Logging.lgALL, True, True, True)

' log a message in Sub 'MySubOrFunction'
myLogger.logINFO "This is my message ..", "MySubOrFunction"

Result:
(28.08.2008 10:53:20)[VBALogger::MySubOrFunction]-INFO:  This is my message ..
```

Setting Log Levels:

Log Levels can be set at startup time using vba_log.properties : The properties file must be located in the same directory as the VBA Module containing the LOGGER Class (and the Logging Module). When the Logger Class is initialized it will look for settings in the 'vba_log.properties' file. Here an Example:

```
#
# -- settings for VBA logging --
#
# LOG_LEVEL:
#
#  DISABLED
#  BASIC 'like Debug.Print
#  FATAL
#  WARN
#  INFO
#  FINE
#  FINER
#  FINEST
#  ALL
#
LOG_LEVEL = info
LOG_TO_CONSOLE = True
LOG_TO_BUFFER = True
LOG_TO_FILE = True
#  Default LOG_FILE_PATH is the same place as VBA project file containing the Logger Modul
#LOG_FILE_PATH=C:¥vba_logger.log
#
```

Log Levels can also be set or changed inside VBA code using the method:

```
Call Logging.setLoggigParams(Logging.lgBASIC, True, True, False)
```

Installing the Logging Framework

There are two options installing the VBA Logging Framework:

- Importing the source moduls into your VBA Projects
- Installation as a xla library

Importing the source moduls

Use your VBA IDE (e.g. Excel (or Word) ->Macros->'Visual Basic Editor') select your VBA Project and use 'Import file..' to import the src files into your project:

- Logging.bas
- Logger.cls
- Logbuffer.cls
- Unit Tests (Optional)

I personally recomend to import the modul files directly into your project. Importing the modules does not create any dependancies and your project is 'redistributable'.

Installation as a XLA library

To install the 'Logging.xla', copy Logging.xla into the MS Office Macros directory** (e.g. 'C:¥Programme¥Microsoft Office¥Office¥Makro') Note that you have to create a Reference to the 'Logging.xla' file to call the Logger from your VBA Project.

To add a Reference to a libray you ususally need to run a method like the following example:

```
Public Sub addLoggerReference()

  On Error GoTo Errhandler:

  Dim location As String
  Dim RefFile As String
  Dim path

  RefFile = "Logging.xla"
  location = getParentFolder(Application.VBE.ActiveVBProject.Filename)
  path = location & "¥" & RefFile

  'add the reference
  Debug.Print "Adding Reference: " & path
  Application.VBE.ActiveVBProject.References.AddFromFile path

  Exit Sub
Errhandler:
   Debug.Print Err.Description
End Sub
```

Your only have to run adding reference code once for your VBA Project: The VBA Project will remember the reference.

**The exact location is dependent on your Office Version and MS Windows Enviroment.

Running the Unit Tests

To run the Unit Test:

Import TestLogging.bas
Run the test calling the Macro 'Test'

Disclaimer:

The following development tool is free to use and change 'As Is' with no guarantee from the author(s). The tools come directly from our own development, so they my not fit 100% but hopefully provide a basis to build on.

TestLogging.bas:

```
Attribute VB_Name = "TestLogging"
'''
''' Basic test macro to test Logger Class using Logging
'''

'define 'myLogger' as 'Object' (not 'Logger') to ensure that
'this test Class works in VBAProjects that reference 'Logging.xla'
'since Public Class Moduls may not be exposed as Type between VBAProjects
Dim myLogger As Object


Sub Test()

 Logging.setModulName (Application.VBE.ActiveVBProject.name)
 Logging.logINFO ("***Starting Logger test..")

 Call printLogLevels

 Logging.log ("***Testing LogLevels..")
 Call Logging.setLoggigParams(Logging.lgALL, True, True, True)
 Call printLogLevels
 Call Logging.setLoggigParams(Logging.lgFINEST, True, True, True)
 Call printLogLevels
 Call Logging.setLoggigParams(Logging.lgFINER, True, True, True)
 Call printLogLevels
 Call Logging.setLoggigParams(Logging.lgFINE, True, True, True)
 Call printLogLevels
 Call Logging.setLoggigParams(Logging.lgINFO, True, True, True)
 Call printLogLevels
 Call Logging.setLoggigParams(Logging.lgWARN, True, True, True)
 Call printLogLevels
 Call Logging.setLoggigParams(Logging.lgFATAL, True, True, True)
 Call printLogLevels
 Call Logging.setLoggigParams(Logging.lgBASIC, True, True, True)
 Call printLogLevels
 Logging.log ("***Now Turn logging off ..")
 Call Logging.setLoggigParams(Logging.lgDISABLED, True, True, True)
 Call printLogLevels
 Call Logging.setLoggigParams(Logging.lgALL, True, True, True)
 Call Logging.log("***Testing logging with 'logpoint' entry ..")
 Call printLogLevelsWithLogPoint

 Call Logging.setLoggigParams(Logging.lgALL, True, False, False)
 Logging.log ("***Testing logBuffer ..")
 Logging.log "----Printing Logging.getLogBuffer to Console only ----"
 Logging.log Logging.getLogBuffer
 Call Logging.setLoggigParams(Logging.lgALL, True, True, True)

 Logging.setModulName ("")

 Call TestLoggerInstance

 Logging.log ("***Testing writing logBuffer to Tracefile ..")
 Logging.writeLogBufferToTraceFile

 Logging.log ("***Testing done.***")

End Sub


Private Sub printLogLevels()
 Logging.log ("-LogBasic = like Debug.Print-")
 Logging.logINFO ("-logINFO-")
 Logging.logWARN ("-logWARN-")
 Logging.logFATAL ("-logFATAL-")
 Logging.logFINE ("-logFINE-")
 Logging.logFINER ("-logFINER-")
 Logging.logFINEST ("-logFINEST-")
End Sub

Private Sub printLogLevelsWithLogPoint()
 Logging.log ("-LogBasic = like Debug.Print-")
 Logging.logINFO "-logINFO-", "printLogLevelsWithLogPoint"
 Logging.logWARN "-logWARN-", "printLogLevelsWithLogPoint"
 Logging.logFATAL "-logFATAL-", "printLogLevelsWithLogPoint"
 Logging.logFINE "-logFINE-", "printLogLevelsWithLogPoint"
 Logging.logFINER "-logFINER-", "printLogLevelsWithLogPoint"
 Logging.logFINEST "-logFINEST-", "printLogLevelsWithLogPoint"
End Sub
```

```
Sub TestLoggerInstance()

  Set myLogger = Logging.getNewLogger(Application.VBE.ActiveVBProject.name)
  Call myLogger.setLoggigParams(Logging.lgALL, True, True, True)
  myLogger.logBASIC "***Starting TestLoggerInstance test.."
  myLogger.logBASIC "-LogBasic = like Debug.Print-", "TestLoggerInstance"
  myLogger.logINFO "-logINFO-", "TestLoggerInstance"
  myLogger.logWARN "-logWARN-", "TestLoggerInstance"
  myLogger.logFATAL "-logFATAL-", "TestLoggerInstance"
  myLogger.logFINE "-logFINE-", "TestLoggerInstance"
  myLogger.logFINER "-logFINER-", "TestLoggerInstance"
  myLogger.logFINEST "-logFINEST-", "TestLoggerInstance"

  'call a sub
  Call MySubOrFunction

  Call myLogger.setLoggigParams(Logging.lgALL, True, False, False)
  myLogger.logBASIC "*** printing the TestLoggerInstance buffer to Console.."
  myLogger.logBASIC myLogger.getLogBuffer

End Sub

Sub MySubOrFunction()
  myLogger.logINFO "This is my message ..", "MySubOrFunction"               ' log a message in Sub 'MySubOrFunction'
End Sub
```

Logging.bas:

```
Attribute VB_Name = "Logging"
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''' Contents:   Logging Modul for VBA - uses 'LOGGER' Class
'''
''' Comments:   Facade for Logger, with static reference to a Logger instance
'''             The Static Logger allows to write log statments to a logbuffer
'''             that can be read for example inside Errorhandling
'''
''' Example: Replacing Debug.Print:
'''     if you use the Logging Module no initialization needs to be done:
'''     instead 'Debug.Print txt' use: 'Logging.log (txt)'
'''
''' Example: Log4VBA sytle logging:
'''
'''     Dim myLogger As Object 'globaly define
'''
'''     Set myLogger = Logging.getNewLogger(Application.VBE.ActiveVBProject.name) ' initalize Logger and set Module Name for example 'VBALogger'
'''     Call myLogger.setLoggigParams(Logging.lgALL, True, True, True)            ' log  ALL to Console, Buffer, File
'''
'''     myLogger.logINFO "This is my message ..", "MySubOrFunction"               ' log a message in Sub 'MySubOrFunction'
'''
'''     Result:
'''     (28.08.2008 10:53:20)[VBALogger::MySubOrFunction]-INFO:  This is my message ..
'''
''' Changing Settings:
'''
''' The bestway to change Loglevels and the settings logging to console, buffer, or logfile
''' is by changing the settings via properties file "vba_log.properties"
''' With this version the properties file is expected in the same directory as the Module
''' containing the LOGGER Class (and the Logging Module)
''' Example:
''' ----------------------------------------------
''' #
''' # -- settings for VBA logging --
''' #
''' # LOG_LEVEL:
''' #
''' #  DISABLED
''' #  BASIC 'like Debug.Print
''' #  FATAL
''' #  WARN
''' #  INFO
''' #  FINE
''' #  FINER
''' #  FINEST
''' #  ALL
''' #
''' LOG_LEVEL = info
''' LOG_TO_CONSOLE = True
''' LOG_TO_BUFFER = True
''' LOG_TO_FILE = True
''' #  Default LOG_FILE_PATH is the same place as Project File containing the Logger Modul
''' #LOG_FILE_PATH=C:Wvba_logger.log
''' ----------------------------------------------
'''
''' Settings can be changed using vba code with the setLoggigParams(..) procedure
''' example:
'''       Call Logging.setLoggigParams(Logging.lgBASIC, True, True, False)
'''
''' Example use for LogBuffer:
'''       If (Err) Then Logging.writeLogBufferToTraceFile
'''
'''
''' Date        Developer               Action
''' ------------------------------------------------------------------------
''' 28/08/08    Christian Bolterauer    Created
'''
'''

Option Explicit

' global to allow access to Logger Class instance via Logging Module
Public defaultLogger As Logger
```

```
'copy of levels from Logger Class to expose levels via the Logging Module
'Note that the enum 'LogLEVEL' is only visable within the VBAProject that contains the Logger Class.
'The Const variables are visable to every Modul where Logging can be accessed
Public Const lgDISABLED = LogLEVEL.DISABLED
Public Const lgBASIC = LogLEVEL.BASIC
Public Const lgFATAL = LogLEVEL.FATAL
Public Const lgWARN = LogLEVEL.WARN
Public Const lgINFO = LogLEVEL.INFO
Public Const lgFINE = LogLEVEL.FINE
Public Const lgFINER = LogLEVEL.FINER
Public Const lgFINEST = LogLEVEL.FINEST
Public Const lgALL = LogLEVEL.ALL

'setter for prime logparameters
Sub setLoggigParams(myloglevel As Integer, toConsole As Boolean, toBuffer As Boolean, toLogFile As Boolean)
  If (myloglevel = LogLEVEL.DISABLED) Then Debug.Print "Logging is disabled."

  'Important: initilaze logger by calling log() before setting params
  log ("Logging with logLevel=" & defaultLogger.getLogLevelName(myloglevel) & " ToConsole=" & toConsole & " ToBuffer=" & toBuffer & " ToLogFile=" & toLo
  Call defaultLogger.setLoggigParams(myloglevel, toConsole, toBuffer, toLogFile)

  'Inital LogfilePath set here
  'Call defaultLogger.setLogFile(Application.ActiveWorkbook.path & "Wvba_logger.log")

End Sub
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Static defaultLogger instance
'
' The live time of this logger instance is as long as the application runs
' This allows to write log messages to a buffer that can be processed even if modules are changed
'
' The defaultLogger is initialized the first time when any of the following log statements is called
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Private Sub thislog(msg As String, myloglevel As LogLEVEL, Optional slogpoint As String)
  Static mydefaultLogger As New Logger 'singelton

  '- if static value is not set assume start of vba session and delete the log file -
  If (defaultLogger Is Nothing) Then
     Call mydefaultLogger.deleteLogFile
  End If

  Call mydefaultLogger.log(msg, myloglevel, slogpoint)
  Set defaultLogger = mydefaultLogger  'refence to static object
End Sub

Public Sub log(sLogText As String, Optional slogpoint As String)
   Call thislog(sLogText, LogLEVEL.BASIC, slogpoint)
End Sub
Public Sub logINFO(sLogText As String, Optional slogpoint As String)
   Call thislog(sLogText, LogLEVEL.INFO, slogpoint)
End Sub
Public Sub logWARN(sLogText As String, Optional slogpoint As String)
   Call thislog(sLogText, LogLEVEL.WARN, slogpoint)
End Sub
Public Sub logFATAL(sLogText As String, Optional slogpoint As String)
   Call thislog(sLogText, LogLEVEL.FATAL, slogpoint)
End Sub
Public Sub logFINE(sLogText As String, Optional slogpoint As String)
   Call thislog(sLogText, LogLEVEL.FINE, slogpoint)
End Sub
Public Sub logFINER(sLogText As String, Optional slogpoint As String)
   Call thislog(sLogText, LogLEVEL.FINER, slogpoint)
End Sub
Public Sub logFINEST(sLogText As String, Optional slogpoint As String)
   Call thislog(sLogText, LogLEVEL.FINEST, slogpoint)
End Sub
Function getLogBuffer()
   If (defaultLogger Is Nothing) Then
      'initilize defaultLogger calling ..
      Call thislog("Retrieving LogBuffer..", LogLEVEL.FINE)
   End If
   getLogBuffer = defaultLogger.getLogBuffer
End Function
'set setModulName: ensures that defaultLogger is initalized before value is set
Public Sub setModulName(myModulName As String)
    If (defaultLogger Is Nothing) Then
      'initilize defaultLogger calling ..
      Call thislog("Setting ModulName to " & myModulName, LogLEVEL.FINE)
    End If
    defaultLogger.ModulName = myModulName
End Sub
Public Sub writeLogBufferToTraceFile(Optional myfilePath As String)
    If (defaultLogger Is Nothing) Then
      'initilize defaultLogger calling ..
      Call thislog("Writing LogBuffer to TraceFile ..", LogLEVEL.FINE)
   End If
   defaultLogger.writeLogBufferToTraceFile (myfilePath)
End Sub

'*********************************************************************************************
'* MODULE:    getNewLogger
'*
'* PURPOSE:   Return a logger object with the defaults set.
'*            The Log Buffer of the new Logger created by this factory method is set
'*            to defaultLogger.strLogbuffer so that all log entries of a session can be traced
'*
'* PARAMETERS: sModulName - the VBA Module that will be used as an identifier within the log file.
'*********************************************************************************************
Public Static Function getNewLogger(sModulName As String) As Logger
    Dim myLogger As New Logger
    myLogger.ModulName = sModulName

    'set the logBuffer to defaultLogger Logbuffer so that all log entries of a session can be traced
```

```
    Set myLogger.cLogbuffer = defaultLogger.cLogbuffer

    Set getNewLogger = myLogger
End Function
```

Logger.cls:

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "Logger"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''' CLASS_MODULE:   VBA Logger - allows 'log4VBA' style Logging in VBA
'''                 - please see the 'Logging' Module for Usage: the Logging Module
'''                   automatically creates a 'Logger' instance and provides additional
'''                   Features
'''
'''                 - use Macro 'Test' from 'TestLogging' for testing and as an example
'''
''' Date            Developer               Action
''' ------------------------------------------------------------------------
''' 28/08/08        Christian Bolterauer    Created
'''

Public cLogbuffer As Logbuffer

Public iLogLevel As Integer
Public bUseLogPrefix As Boolean
Public bConsole As Boolean
Public bBuffer As Boolean
Public bToLogFile As Boolean
Public LogFilePath As String
Public TraceFilePath As String
Public bDelLogFileAtSetup As Boolean
Public PropsFileName As String

'ModulName
Public ModulName As String

'Define log levels
Public Enum LogLEVEL
  DISABLED = 0
  BASIC = 1 'like Debug.Print
  FATAL = 2
  WARN = 3
  INFO = 4
  FINE = 5
  FINER = 6
  FINEST = 7
  ALL = 8
End Enum

' The defaults
Const DEFAULT_LOG_LEVEL% = LogLEVEL.INFO
Const DEFAULT_LOG_Console = True
Const DEFAULT_LOG_Buffer = False
Const DEFAULT_LOG_FILE = False
Const DEFAULT_PROPSFILE_NAME = "vba_log.properties"
'


'Class Konstructor
Private Sub Class_Initialize()
   On Error GoTo Errhandler:
   Dim localpath As String
   Set cLogbuffer = New Logbuffer
   bUseLogPrefix = True
   bDelLogFileAtSetup = True

   'default
   ModulName = ""

   'set default location of props file to directory of this Logger and add default name
   localpath = getParentFolder(Application.VBE.ActiveVBProject.Filename) 'set path to location of file containing this Logger
   PropsFileName = localpath & "W" & DEFAULT_PROPSFILE_NAME

   'make sure defaults are set
   Call setLoggigParams(DEFAULT_LOG_LEVEL, DEFAULT_LOG_Console, DEFAULT_LOG_Buffer, DEFAULT_LOG_FILE)
   'set default log file path
   LogFilePath = localpath & "W" & "vba_logger.log"
   TraceFilePath = localpath & "W" & "vba_trace.log"
   'check if params can be set from a properties file and overwrite defaults if available
   Call getLogParamsFromFile
   'set log file
   Call setLogFile(LogFilePath, False)

Exit Sub

Errhandler:
  Debug.Print "Error in Logger.Class_Initialize & "; " & Err.Number & "; " & Err.Description"

End Sub

'set logging parameters
Public Sub setLoggigParams(level As Integer, toConsole As Boolean, toBuffer As Boolean, toLogFile As Boolean, Optional deleteExistingLogFile)
    Dim delLogfile As Boolean
```

```
      iLogLevel = level
      bConsole = toConsole
      bBuffer = toBuffer
      bToLogFile = toLogFile

      If IsMissing(deleteExistingLogFile) Then
         delLogfile = False
      Else
         delLogfile = deleteExistingLogFile
      End If
      ' delete currently set Logfile if set
      If (delLogfile) Then deleteLogFile


End Sub

'The main log procedure
Public Sub log(sLogText As String, level As LogLEVEL, Optional slogpoint As String)

    If (Me.iLogLevel > LogLEVEL.DISABLED And Me.iLogLevel >= level) Then
        If IsMissing(slogpoint) Then
           Call WriteLog(sLogText, level, "")
        Else
           Call WriteLog(sLogText, level, slogpoint)
        End If
    End If
End Sub

Public Sub logBASIC(sLogText As String, Optional slogpoint As String)
   Call Me.log(sLogText, LogLEVEL.BASIC, slogpoint)
End Sub
Public Sub logINFO(sLogText As String, Optional slogpoint As String)
   Call Me.log(sLogText, LogLEVEL.INFO, slogpoint)
End Sub
Public Sub logWARN(sLogText As String, Optional slogpoint As String)
   Call Me.log(sLogText, LogLEVEL.WARN, slogpoint)
End Sub
Public Sub logFATAL(sLogText As String, Optional slogpoint As String)
   Call Me.log(sLogText, LogLEVEL.FATAL, slogpoint)
End Sub
Public Sub logFINE(sLogText As String, Optional slogpoint As String)
   Call Me.log(sLogText, LogLEVEL.FINE, slogpoint)
End Sub
Public Sub logFINER(sLogText As String, Optional slogpoint As String)
   Call Me.log(sLogText, LogLEVEL.FINER, slogpoint)
End Sub
Public Sub logFINEST(sLogText As String, Optional slogpoint As String)
   Call Me.log(sLogText, LogLEVEL.FINEST, slogpoint)
End Sub

Private Sub WriteLog(sLogText, level As LogLEVEL, slogpoint As String)

  Dim LogMessage As String
  Dim sDateTime As String
  Dim sLogPrefix As String

  LogMessage = getLogPrefix(level, slogpoint) & sLogText

  ' write to console
  If Me.bConsole Then Debug.Print (LogMessage)
  ' write to Buffer
  If Me.bBuffer Then cLogbuffer.addline (LogMessage)
  ' write to file
  If Me.bToLogFile Then writeToLogFile (LogMessage)

End Sub

' get LogLevelName for Integer value
Public Function getLogLevelName(level As Integer)
    Dim myLevelName As String

    Select Case level
    Case LogLEVEL.DISABLED:
       myLevelName = "DISABLED"
    Case LogLEVEL.BASIC:
       myLevelName = "BASIC"
    Case LogLEVEL.INFO:
       myLevelName = "INFO:"
    Case LogLEVEL.WARN:
       myLevelName = "WARN:"
    Case LogLEVEL.FATAL:
       myLevelName = "FATAL:"
    Case LogLEVEL.FINE:
       myLevelName = "FINE:"
    Case LogLEVEL.FINER:
       myLevelName = "FINER:"
    Case LogLEVEL.FINEST:
       myLevelName = "FINEST:"
    Case LogLEVEL.ALL:
       myLevelName = "ALL:"

    Case Else
       myLevelName = "Level is not defined:"
  End Select

  getLogLevelName = myLevelName

End Function

Private Function getLogPrefix(level As LogLEVEL, logpoint As String)
  Dim sDateTime As String
  Dim myLevelPrefix As String
  Dim mySubModul As String
```

```
    Dim iLevel As Integer

    If Not (bUseLogPrefix) Or level = LogLEVEL.BASIC Then   'when level = LogLEVEL.BASIC no prefix to simulate Debug.Print
       getLogPrefix = ""
       Exit Function
    End If

    iLevel = level ' to Integer
    myLevelPrefix = getLogLevelName(iLevel)

    If (Len(Me.ModulName) > 0 And Len(logpoint) > 0) Then
       mySubModul = "[" & Me.ModulName & "::" & logpoint & "]"
    ElseIf (Len(logpoint) > 0) Then
       mySubModul = "[" & logpoint & "]"
    ElseIf (Len(Me.ModulName) > 0) Then
       mySubModul = "[" & Me.ModulName & "]"
    Else
       mySubModul = ""
    End If

    sDateTime = CStr(Now())
    'ToDo provide different output styles ..
    'getLogPrefix = myLevelPrefix & " (" & sDateTime & ") - "
    getLogPrefix = "(" & sDateTime & ")" & mySubModul & "-" & myLevelPrefix & "  "

  End Function

  Private Sub writeToLogFile(logmsg As String)
      On Error GoTo Errhandler:
      If Len(Me.LogFilePath) = 0 Then
         Debug.Print "Error: Log file path is empty."
         Exit Sub
      End If

      Dim FileNum As Integer
      FileNum = FreeFile ' next file number
      Open Me.LogFilePath For Append As #FileNum ' creates the file if it doesn't exist
      Print #FileNum, logmsg ' write information at the end of the text file
      Close #FileNum ' close the file
  Exit Sub

  Errhandler:
      Debug.Print "Error writing to Logfile: " & Me.LogFilePath & " " & Err.Number & " " & Err.Description

  End Sub

  Public Sub writeLogBufferToTraceFile(Optional myfilePath As String)
      On Error GoTo Errhandler:
      Dim mytracefile As String

      If Len(myfilePath) = 0 Then
          mytracefile = Me.TraceFilePath
      Else
          mytracefile = myfilePath
      End If
      If Len(mytracefile) = 0 Then
         Me.logFATAL "Error: Trace file path is empty."
         Exit Sub
      End If
      'write to trace file
      Me.cLogbuffer.writeLogBufferToTraceFile (mytracefile)
  Exit Sub

  Errhandler:
      Debug.Print "Error writing to Tracefile: " & mytracefile & " " & Err.Number & " " & Err.Description
  End Sub

  Private Sub readPropertiesFile(path As String)

      On Error GoTo Errhandler:
      Dim txtline As String
      Dim para() As String
      Dim mymsg As String

      If Len(path) = 0 Then GoTo Errhandler
      Open path For Input As #1    ' open file
      Do While Not EOF(1)          ' Loop until end of file
        Line Input #1, txtline     ' read line

        'Debug.Print txtline 'test
        para = readParameter(txtline)

        If Len(para(0)) = 0 Then
          'continue
        ElseIf ("LOG_LEVEL" = UCase(para(0))) Then
          Call setLogLevel(para(1))
        ElseIf ("LOG_TO_CONSOLE" = UCase(para(0))) Then
          bConsole = valIsTrue(para(1))
        ElseIf ("LOG_TO_BUFFER" = UCase(para(0))) Then
          bBuffer = valIsTrue(para(1))
        ElseIf ("LOG_TO_FILE" = UCase(para(0))) Then
          bToLogFile = valIsTrue(para(1))
        ElseIf ("LOG_FILE_PATH" = UCase(para(0))) Then
          Me.LogFilePath = para(1)
        End If
      Loop
      Close #1
      'show settings
      mymsg = "Logging with logLevel=" & getLogLevelName(iLogLevel) & " toConsole=" & bConsole & " toBuffer=" & bBuffer & " toLogFile=" & bToLogFile
      Call log(mymsg, LogLEVEL.BASIC)
  Exit Sub

  Errhandler:
      Debug.Print "Error reading Properties File: " & path & " " & Err.Number & " " & Err.Description
```

```
End Sub
'delete log file currently set
Public Sub deleteLogFile()
  On Error GoTo Errhandler:
  If (FileExists(Me.LogFilePath)) Then
     Kill (Me.LogFilePath)
  End If
Exit Sub

Errhandler:
  Debug.Print "Error deleting Logfile " & Me.LogFilePath & " " & Err.Number & " " & Err.Description

End Sub
'set logfilepath
'- will delete an existing log file if bDelLogFileAtSetup is set to true
Public Sub setLogFile(filePath As String, delExitingFile As Boolean)
  On Error GoTo Errhandler:

  Me.LogFilePath = filePath
  'delete if set to true
  If (delExitingFile) Then Call deleteLogFile
  If (bToLogFile) Then Debug.Print "Logfile set to: " & LogFilePath
Exit Sub

Errhandler:
  Debug.Print "Error setLogFile " & LogFilePath & " " & Err.Number & " " & Err.Description

End Sub

Public Function getLogParamsFromFile()

  On Error GoTo Errhandler:
  If (FileExists(PropsFileName)) Then
     Debug.Print "Reading: " & PropsFileName
     'read and set parameter from properties file
     readPropertiesFile (PropsFileName)
     getLogParamsFromFile = True
   Exit Function
  End If
  getLogParamsFromFile = False

Exit Function
Errhandler:
  Debug.Print "Error getLogParamsFromFile " & PropsFileName & " " & Err.Number & " " & Err.Description
  getLogParamsFromFile = False
End Function

Private Sub setLogLevel(level As String)
  Dim mylevel
  mylevel = UCase(level)

  Select Case mylevel
    Case "DISABLED":
       iLogLevel = LogLEVEL.DISABLED
    Case "BASIC":
       iLogLevel = LogLEVEL.BASIC
    Case "INFO":
       iLogLevel = LogLEVEL.INFO
    Case "WARN":
       iLogLevel = LogLEVEL.WARN
    Case "FATAL":
       iLogLevel = LogLEVEL.FATAL
    Case "FINE":
       iLogLevel = LogLEVEL.FINE
    Case "FINER":
       iLogLevel = LogLEVEL.FINER
    Case "FINEST":
       iLogLevel = LogLEVEL.FINEST
    Case "ALL":
       iLogLevel = LogLEVEL.ALL
  End Select
End Sub

Public Function getLogBuffer() As String
   getLogBuffer = cLogbuffer.strLogbuffer
End Function


'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''' Utils
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'-- extract full path of parent folder of file mypath
Public Function getParentFolder(mypath As String) As String
  Dim pos As Integer
  Dim fullpath As String

  pos = InStrRev(mypath, "W")
  If (pos <> 0) Then
    getParentFolder = Left(mypath, pos - 1)
    Exit Function
  End If
  getParentFolder = ""
End Function

'-- Check File Exists --
Public Function FileExists(path As String) As Boolean
  FileExists = (Dir(path) <> "")
End Function

'-- isNothing --
Function checkIsNothing(obj As Object)
  If (obj Is Nothing) Then
     checkIsNothing = True
```

```
    Else
       checkIsNothing = False
    End If
  End Function

  '-- parameters ---
  Function readParameter(line As String) As String()
    Dim txtarr() As String
    Dim proparray(2) As String

    txtarr = VBA.Split(line, "=")
    If (UBound(txtarr) > 0) Then
      proparray(0) = VBA.Trim(txtarr(0))
      proparray(1) = VBA.Trim(txtarr(1))
      readParameter = proparray
    Else
      readParameter = proparray
    End If
  End Function

  '-- check text coded boolean value (if read from text file) --
  Function vallsTrue(boolval As String) As Boolean
      If ("TRUE" = VBA.UCase(boolval)) Then
          vallsTrue = True
          Exit Function
      End If
      vallsTrue = False
  End Function
```

Logbuffer.cls:

```
  rel=

  VERSION 1.0 CLASS
  BEGIN
    MultiUse = -1  'True
  END
  Attribute VB_Name = "Logbuffer"
  Attribute VB_GlobalNameSpace = False
  Attribute VB_Creatable = False
  Attribute VB_PredeclaredId = False
  Attribute VB_Exposed = False

  Option Explicit

  '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
  ''' CLASS_MODULE:   VBA Logbuffer - allows a Reference e.g. the 'Set' Method on a String Property
  '''
  '''                 - see the 'Logging' Module for Usage: the Logging Module
  '''                   automatically creates a 'Logger' instance and provides additional
  '''                   Features
  '''
  '''                 - use Macro 'Test' from 'TestLogging' for testing and as an example
  '''
  ''' Date            Developer               Action
  ''' ------------------------------------------------------------------------
  ''' 28/08/08        Christian Bolterauer    Created
  '''


  Public strLogbuffer As String

  Private Sub Class_Initialize()
     strLogbuffer = ""
  End Sub

  Public Sub addline(logmsg As String)
     If (Len(strLogbuffer) > 0) Then
       strLogbuffer = strLogbuffer & vbLf & logmsg
     Else
       strLogbuffer = logmsg  'avoid empty line when strLogbuffer=""
     End If
  End Sub

  Public Sub writeLogBufferToTraceFile(myfilePath As String)

      On Error GoTo Errhandler:
      Dim lines() As String
      Dim line As Variant

      If Len(myfilePath) = 0 Then
         Debug.Print "Error: Trace file path is empty."
         Exit Sub
      End If

      Dim FileNum As Integer
      FileNum = FreeFile ' next file number
      Open myfilePath For Output As #FileNum ' creates the file if it doesn't exist
      lines = VBA.Split(Me.strLogbuffer, VBA.vbLf)
      For Each line In lines
         Print #FileNum, line ' write Logbuffer to text file
      Next line
      Close #FileNum ' close the file
  Exit Sub

  Errhandler:
      Debug.Print "Error writing to Tracefile: " & myfilePath & " " & Err.Number & " " & Err.Description

  End Sub
```

Comments, suggestions please contact me at csinfo@bolterauer.de

---