19.5 Logging

Die Komponente *gb.logging* implementiert ein flexibles *System für die Protokollierung* von Gambas-Anwendungen. Diese Komponente stellt ihre Funktionalität über die beiden Klassen *Formatter* und *Logger* zur Verfügung.

- Die Klasse *Formatter* bietet eine gute Möglichkeit für eine effiziente Fehlersuche in der Testphase eines Programms.
- Die komplette Log-Funktionalität erreichen Sie durch die Verwendung der Klasse Logger (Logger-Objekt).
- Die Formatierung der Protokoll-Texte basiert auf dem Dokument *RFC 5424*, in dem das Standard-Format *(→ http://tools.ietf.org/html/rfc5424)* für System-Log-Dateien (syslog) beschrieben wird.

19.5.1 Klasse Formatter

Die Klasse *Formatter (gb.logging)* kann wie eine Funktion benutzt werden. Diese Klasse formatiert eine Zeile, um die erforderlichen Protokoll-Daten automatisch zu ersetzen. Welche Format-Tags Sie einsetzen können, finden Sie im Abschnitt 19.5.2.2.

In der Klasse *Formatter* können Sie – bis Gambas 3.5.3 – diese vier 4 Format-Tags *nicht* einsetzen:

```
$(callLocation); $(callFile); $(callLine) und $(callFunction)
```

Beispiel:

Die folgende Anweisung im Quelltext bestimmt einerseits den Protokoll-Text und andererseits das Ausgabe-Format sowie das Log-Level, das dem Protokoll-Text zugewiesen wird (→ Abschnitt 19.5.2.1 Level-Konstanten und 19.5.2.2 (Eigenschaft Level)):

```
Print Formatter(Text-String, Format-String, Log-Level)
Print Formatter("Test-Mitteilung", "[$(time)] - $(message) - $(levelname)", Logger.Warning)(*)
```

Sie erzeugt zur Laufzeit folgende Ausgabe in der Gambas-IDE:

```
[18:59:04.227] - Test-Mitteilung - WARNING
```

Sie können die o.a. Anweisung (*) mit Ihren speziellen Protokoll-Texten an unterschiedlichen Stellen in den Quelltext einfügen.

Mit gleichem Effekt können Sie auch nachstehende Anweisung aus dem Bereich 'Error-Management' einsetzen:

```
Debug "[" & Format(Time(), "hh:nn:ss.uu") & "]" & " - Test-Mitteilung" & " - WARNUNG"
```

die folgende Ausgabe in die Konsole der IDE schreibt, die zusätzlich noch Angaben zur Datei, zum Event und zur Angabe der Zeile im Quelltext enthält:

```
FMain.btnDebug_Click.11: [18:59:04.227] - Test-Mitteilung - WARNUNG
```

19.5.2 Klasse Logger

Die Klasse *Logger (gb.logging)* stellt u.a. grundlegende Mechanismen zum Senden von Protokoll-Texten zur Verfügung. Sie können so viele Logger-Objekte erstellen wie Sie benötigen, denn die Komponente *gb.logging* ist hervorragend geeignet, um zum Beispiel mehrere Protokolle für unterschiedliche Log-Level zu verwalten.

Die Klasse *Logger* verfügt über die im nächsten Abschnitt vorgestellten

- fünf Konstanten,
- zwei Eigenschaften und
- eine Methode.

19.5.2.1 Konstanten

Konstante	Numerischer Wert	Beschreibung
Critical	0	Die Level-Konstante definiert einen Protokoll-Text als kritische Meldung.
Error	1	Die Level-Konstante definiert einen Protokoll-Text als Fehler-Meldung.
Warning	2	Die Level-Konstante definiert einen Protokoll-Text als Warnung.
Info	3	Die Level-Konstante definiert einen Protokoll-Text als Informationsmeldung.
Debug	4	Die Level-Konstante definiert einen Protokoll-Text als Debug-Meldung.

Tabelle 19.5.2.1.1: Übersicht zu den Level-Konstanten der Klasse Logger

19.5.2.2 Eigenschaften

Eigenschaft	Datentyp	Default	Beschreibung
Format	String	*	Setzt das Format für die Protokoll-Texte (Logger-Objekt) oder gibt das Format zurück.
Level	Integer	3	Setzt das minimale Log-Level für ein Logger-Objekt oder gibt dieses zurück. Die Werte von *Level* liegen im Intervall von 0 bis 4 → Konstanten.

Tabelle 19.5.2.2.1: Eigenschaften der Klasse Logger

Das Standard-Format * der Klasse Logger ist:

```
[$(now)] [$(levelname)] [$(callLocation)] $(message)
```

und muss nur dann von Ihnen neu festgelegt werden (→ Tabelle 19.5.2.1.1), wenn Sie ein anderes Protokoll-Text-Format einstellen wollen.

Hinweise zum Format:

Ein Format-String beschreibt über die Tags $(..), welche *Informationen* an welcher *Position* in die Log-Nachricht aufgenommen werden:

```
$(message): Fügt den Mitteilungstext ein.
$(now): Fügt das Systemdatum und die Systemzeit ein.
$(date): Fügt das aktuelle System-Datum ein.
$(time): Fügt die aktuelle System-Zeit ein.

$(callLocation): Äquivalent zu $(callFile).$(callFunction)$(callLine).
$(callFile): Fügt den Namen der Quelltext-Datei für die Methode ein, die die Log-Nachricht aufgerufen hat.
$(callLine): Fügt den Namen der Quelltext-Zeile für die Methode ein, die die Log-Nachricht aufgerufen hat.
$(callFunction): Fügt den Namen der Funktion ein, die die Log-Nachricht aufgerufen hat.

$(ptimer): Fügt den Wert des Prozess-Timers ein.
$(ptimerint): Fügt den Wert des Prozess-Timers in Sekunden ein.

$(levelno): Fügt den Protokoll-Text-Level als ganze Zahl ein.
$(levelname): Fügt den Protokoll-Text-Level als String-Repräsentation des Levels ein.
$(version): Fügt die Version der Anwendung ein.
$(gbversion): Fügt die Version der Gambas-Runtime ein.
$(host): Fügt den Hostname des Systems ein, auf dem das Gambas-Programm läuft
$(pid): Fügt die Prozess-ID des Prozesses ein, der den Protokoll-Eintrag auslöste.
```

- Beachten Sie die Schreibweise der o.a. Tags im Format-String (→ *case-sensitiv)* !
- Für die Eigenschaft *Level* setzen Sie entweder die Konstanten ein oder deren numerische Werte, wobei den Konstanten der Vorzug zu geben ist.
- Sie können auch eigenen Text oder einzelne Zeichen in den Format-String einfügen.

19.5.2.3 Methode

Die Klasse *Logger* verfügt nur über die Methode *Logger.isEnabledFor(..)*:

```
Funktion isEnabledFor(iLevel As Integer) As Boolean
```

Die Funktion gibt *True* zurück, wenn das aktive Logger-Objekt 'MyLog' für das durch 'iLevel' angegebene Protokoll-Level aktiviert wurde.

Für die *Demonstration* der Funktion können Sie diesen Quelltext nutzen, um zu prüfen, wie viel und welche Protokoll-Level für 'MyLog' aktiv sind:

```
MyLog.Level = Logger.Error
If MyLog.isEnabledFor(Logger.Critical) Then Print "Critical-Level aktiv!"
If MyLog.isEnabledFor(Logger.Error) Then Print "Error-Level aktiv!"
If MyLog.isEnabledFor(Logger.Warning) Then Print "Warning-Level aktiv!"
If MyLog.isEnabledFor(Logger.Info) Then Print "Info-Level aktiv!" ' Default: Logger.Info (=3)
If MyLog.isEnabledFor(Logger.Debug) Then Print "Debug-Level aktiv"
```

Die Anzeige in der Konsole zeigt folgende zwei Level an, weil mit 'MyLog.Level = Logger.Error' nur die *Log-Level* ≤ *Logger.Error* aktiv sind:

```
Critical-Level aktiv!
Error-Level aktiv!
```

19.5.2.4 Logger-Objekt

Ein Logger-Objekt können Sie so anlegen:

```
Public hLogger = Logger
hLogger = Logger ( [ iMinLevel As Integer, sOutput As String ] )
```

Beispiele:

(a) Logger-Objekt mit dem Standard-Level (=Logger.Info) zur Anzeige der Protokoll-Texte in der Konsole der IDE
(b) Logger-Objekt mit einem selbst vorgegebenen Level (0..4) zur Anzeige der Protokoll-Texte in der Konsole der IDE.
(c) Logger-Objekt mit einem selbst vorgegebenen Level (0..4) und Speicherung der Protokoll-Texte in einer Log-Datei in einem Verzeichnis, in dem der Benutzer auch Schreibrechte besitzt.

```
(a)    hLogger As New Logger      ' Logger-Level » Info
(b)    hLogger As New Logger(2)   ' Logger-Level » Warning
(c)    hLogger As New Logger(Logger.Error, Lower(User.Home &/ Application.Name & ".log"))
```

19.5.2.5 Protokollierung

Für ein Objekt der Klasse Logger – zum Beispiel mit dem Namen 'MyLog' – gilt:

```
Sub MyLog ( sMessage As String [ , iLevel As Integer ] )
```

mit den zwei Parametern:

- sMessage ist der Text, der nach der vorgegebenen Formatierung ausgegeben wird und
- iLevel bestimmt das Log-Level (→ Tabelle 19.5.2.1.1) und ist ein optionaler Parameter,

um einen Protokoll-Text für das aktive Logger-Objekt 'MyLog' zu erzeugen.

Mit Hilfe der Prozedur *MyLog(..)* und der Verwendung eines *Logger-Objekts* mit seinen Konstanten, Eigenschaften sowie der Methode isEnabledFor(..) mit 'Function isEnabledFor (iLevel As Integer) As Boolean' lässt sich ein flexibles Log-System für ein Gambas-Programm realisieren.

19.5.3 Beispiel

Genauso wie mit der Klasse *Formatter* können Sie auch mit der Klasse *Logger* Protokoll-Texte an die Standard-Ausgabe (→ Konsole) senden und dort anzeigen.

Für die Entwicklung und für den Test von Gambas-Programmen ist es durchaus von Vorteil, wenn Sie die Protokoll-Texte in einer Log-Datei (→ Variante (c)) speichern.

Hier ein Quelltext-Ausschnitt nach Variante (c):

```
[1]  Public MyLog As Logger
[2]  Public FilePath As String = Lower(User.Home &/ Application.Name & ".log")
[3]  Public bAppendMode As Boolean = True ' True → Log-Datei fortschreiben
[4]
[5]  Public Sub Form_Open()
[6]    FMain.Center
[7]    FMain.Resizable = False
[8]  …
[9]  ' Minimales Log-Level, Log-Dateipfad und Modus festlegen
[10]   If Exist(FilePath) Then
[11]      If bAppendMode = False Then
[12]         Try Kill FilePath
[13]         Wait
[14]         MyLog = New Logger(Logger.Error, FilePath)
[15]      Endif ' bAppendMode = False ?
[16]   Endif ' Exist(FilePath) ?
[17]   MyLog = New Logger(Logger.Error, FilePath)
[18]
[19] ' Log-Format festlegen
[20]   MyLog.Format = "$(message)"
[21] ' Log-Text ausgeben
[22]   If bAppendMode = True And Exist(FilePath) Then MyLog("", MyLog.Level) ' Leerzeile einfügen
[23]
[24]   MyLog.Format = "$(message)"
[25]   MyLog(Format(Now, "dddd - dd. mmmm yyyy"), MyLog.Level)
[26]
[27]   MyLog.Format = "$(time)" & gb.Tab & "$(message)" & " mit Log-Level " & " ≤ " & " $(levelname)"
[28]   MyLog("LOG-NEUSTART", MyLog.Level)
[29]
[30]   MyLog.Format = "$(message)"
[31]   MyLog("-------------------------------------------------", MyLog.Level)
[32]
[33] ' Log-Format für alle weiteren Protokoll-Texte festlegen
[34]   MyLog.Format = "$(time)" & gb.Tab & "[$(levelname)] [$(callLocation)]" & " →  " & "$(message)"
[35]
[36] End ' Form_Open()
```

Nun können Sie an den passenden, oft kritischen Stellen im Quelltext mit modifizierten Anweisungen weitere Protokoll-Texte in der Log-Datei protokollieren:

```
MyLog("Eingabefehler Z1", Logger.Warning)                    oder
MyLog("Eingabe-String = " & sInput, Logger.Error)            oder
MyLog("Division durch Null!", Logger.Critical)               oder

MyLog.Level = Logger.Info
MyLog("Änderung Log-Level auf " & MyLog.Level, MyLog.Level)
```

Hier sehen Sie einen Log-Datei-Auszug im Modus 'Append' bei unterschiedlichem Log-Level:

```
Samstag - 18. Januar 2014
15:21:50.803  LOG-NEUSTART mit Log-Level ≤ WARNING
-------------------------------------------------

15:21:54.12   [CRITICAL] [FMain.IsComplex.339] →  Eingabe-String = 3-4i,,
15:21:55.228  [WARNING] [FMain.btnAddieren_Click.139] →  Eingabefehler Z1 oder Z2
15:21:57.9    [ERROR] [FMain.btnConvert_Click.96] →  Eingabefehler Z1
15:21:59.719  [CRITICAL] [FMain.IsComplex.339] →  Eingabe-String = 3-4i,,

Samstag - 18. Januar 2014
15:41:07.429  LOG-NEUSTART mit Log-Level ≤ ERROR
-------------------------------------------------

15:41:14.908  [ERROR] [FMain.btnConvert_Click.96] →  Eingabefehler Z1
15:41:16.282  [CRITICAL] [FMain.IsComplex.339] →  Eingabe-String = 3-4i,
15:41:17.684  [ERROR] [FMain.btnIsComplex_Click.122] →  Eingabefehler Z1
```

Hinweise:

- Beachten Sie, dass nur die Protokoll-Texte in die Log-Datei aufgenommen werden, bei denen das angegebene Log-Level kleiner oder gleich dem aktuellen Log-Level ist!
- Protokoll-Texte mit einem größeren Log-Level werden dann ignoriert. Bei einem Log-Level kleiner oder gleich *Logger.Error* zum Beispiel – wie in der zweiten Sektion im o.a. Log-Datei-Auszug – werden nur Fehlermeldungen und kritische Meldungen protokolliert.
- Geben Sie kein Log-Level vor, dann wird als Log-Level das Standard-Log-Level *Logger.Info* mit dem numerischen Wert 3 gesetzt.

Die Größe der Log-Datei sollten Sie u.U. beschränken, wenn Sie das Protokoll laufend ergänzen wollen. Ein akzeptables Verfahren besteht darin, ab einer bestimmten Datei-Größe den ältesten Log-Eintrag zu löschen, wenn ein neuer dazu kommt. Oder Sie erzeugen ab einer bestimmten Datei-Größe aus der aktuellen Log-Datei ein Archiv und legen eine neue Log-Datei an.

19.5.4 Exkurs

Der Abschnitt *Exkurs* stellt Ihnen zwei Log-Varianten vor, die beide ihre speziellen Aspekte besitzen.

- Variante 1 bietet die Möglichkeit, Protokoll-Texte entweder sofort in der Konsole auszugeben und anzuzeigen und/oder als Meldung in einem eigenen Fenster. Als Besonderheit gilt, dass alle Protokoll-Texte zur Laufzeit des Programms im Speicher gehalten werden (Collection) und erst zum Programmende in einer Log-Datei permanent gespeichert werden könnten.
- Variante 2 ist so ausgelegt, dass *jeder* Protokoll-Text *sofort* in einer Log-Datei speichert wird und bietet damit m.E. einen kleinen Vorteil gegenüber Variante 1, wenn man ein Programm testet.

19.5.4.1 Variante 1

Der folgende Quelltext setzt die Vorgaben der Variante 1 in einem *Modul* 'MyLog' um – jedoch ohne Speicherung in einer Log-Datei:

```
' Gambas module file - Autor: Caveat - gambas@caveat.demon.co.uk

PRIVATE iDebugMode AS Integer
PRIVATE iLogIndex AS Integer
PRIVATE cLogLines AS Collection

PUBLIC CONST NO_DEBUG AS Integer = 0
PUBLIC CONST DEBUG_LOG_ONLY AS Integer = 1
PUBLIC CONST DEBUG_LOG_AND_PRINT AS Integer = 2
PUBLIC CONST DEBUG_MESSAGE_LOG_AND_PRINT AS Integer = 3

Public Sub SetDebugMode(iNewMode As Integer)
  iDebugMode = iNewMode
End ' SetDebugMode(iNewMode AS Integer)

Public Sub LogMessage(sMessage As String, bShowAsError As Boolean)

  If iDebugMode = NO_DEBUG Then Return
  If cLogLines = Null Then
     cLogLines = New Collection
  Endif ' cLogLines = Null

  cLogLines.Add(sMessage, Str(iLogIndex))
  Inc iLogIndex

  If iDebugMode = DEBUG_LOG_AND_PRINT Then
     Print sMessage
  Else If iDebugMode = DEBUG_MESSAGE_LOG_AND_PRINT Then
     Message(sMessage)
     Print sMessage
  Endif ' iDebugMode = DEBUG_LOG_AND_PRINT

  If bShowAsError Then Message.Error(sMessage)' Zusätzliche Anzeige – aber nur bei Fehlern!

End ' LogMessage(sMessage AS String, bShowAsError AS Boolean)


Public Sub DisplayLogLines()
  Dim sLogLine As String

  If cLogLines = Null Then Return
  For Each sLogLine In cLogLines
    Print cLogLines.Key & ". " & sLogLine
  Next ' sLogLine

End ' DisplayLogLines()
```

Ob Sie den Protokoll-Text nur in der Konsole sehen wollen oder auch als Text in einem eigenen Meldungsfenster, legen Sie mit der Prozedur *SetDebugMode(..)* unter Verwendung der definierten Konstanten fest.

Im Haupt-Programm rufen Sie Protokoll-Texte zum Beispiel so auf:

```
MyLogger.SetDebugMode(1) ' DEBUG_LOG_ONLY
...
  Dim hFile As File
  Dim FilePath As String

  FilePath = "/home/hans/Bilder" &/ "gambas.png"

  Try hFile = Open FilePath For Read
  If Error Then
     MyLogger.LogMessage("Datei-Pfad: " & FilePath, True)
     MyLogger.LogMessage("Fehler: " & Error.Text & " @ " & Error.Where, False)
  Endif ' Error ?
```

Die Ausgabe des Log-Inhalts über *DisplayLogLines()* zeigt für den o.a. Quelltextausschnitt:

```
1. Datei-Pfad: /home/hans/Bilder/gambas.png
2. Fehler: Unable to load image @ Stock.LoadIcon.444
```

19.5.4.2 Variante 2

In der Variante 2 werden ausgewählte Programmausgaben – hier sind es Temperaturwerte – in festen Zeitintervallen ausgelesen und in einer Log-Datei protokolliert. Das leistet die angegebene Timer-Prozedur *MyTimer_Timer()* von MyTimer. Sie können bei jedem Programm-Neustart entscheiden, ob die u.U. existierende Log-Datei gelöscht werden soll oder das Protokoll fortgeschrieben wird. Alternativ können Sie zum Beispiel auch eine CheckBox einsetzen, um diese Entscheidung in der Start-Prozedur festzulegen.

Der Quelltext ist übersichtlich und wird in relevanten Auszügen angegeben:

```
Private MyTimer As Timer

Public Sub Form_Open()
...
  MyTimer = New Timer As "MyTimer"
  MyTimer.Delay = 1000 * 120 ' Intervall der Datenspeicherung (→ 2 Minuten)
...
End ' Form_Open()

' Einbau in eine Start-Prozedur:
...
  If Exist(Application.Path &/ "rs232log.txt") Then
     If Message.Question("Letztes Mess-Protokoll löschen?", "Ja - löschen!", "Nein!") = 1 Then
        Try Kill Application.Path &/ "rs232log.txt"
        Wait
     Endif ' Message.Question(..) ?
  Endif ' Exist(Application.Path &/ "rs232.txt") ?

  AddTextToFile("MESS-PROTOKOLL")
  AddTextToFile("DATUM:  " & Format(Now, "dd. mmmm yyyy"))
  AddTextToFile("---------------------------")

  MyTimer.Start
...

Public Sub MyTimer_Timer()
  If RS232.Status = Net.Active Then
     AddTextToFile(">  " & Format(Now, "hh:nn:ss") & "  |  " & "T = " & Asc(sTemperatureValue) & " °C")
  Endif ' RS232.Status = Net.Active ?
End ' MyTimer_Timer()

Public Sub AddTextToFile(Text As String)
  Dim hFile As File
  Dim FilePath As String

  FilePath = Application.Path &/ "rs232log.txt"

  Try hFile = Open FilePath For Append
  If Error Then
     Message.Error("Datei-Fehler")
     Return
  Endif ' Error ?
  Print #hFile, Text
  Close #hFile

End '  AddTextToFile(Text As String)
```

Hier ein Auszug aus einer Log-Datei:

```
MESS-PROTOKOLL
DATUM:  17. Januar 2014
---------------------------
>  09:47:56  |  T = 22 °C
>  09:49:56  |  T = 24 °C
>  09:51:56  |  T = 26 °C
>  09:53:56  |  T = 28 °C
```

Selbstverständlich können Sie auch RS232-Schnittstellen-Parameter oder Fehlermeldungen in der Log-Datei protokollieren.