11.5.1 Fehler-Erkennung und Fehler-Behandlung – Try, Finally und Catch

Besonders bei der Arbeit mit Verzeichnissen und Dateien können Fehler auftreten, die sich schwer vorhersehen lassen. Sie entstehen dadurch, dass sich zum Beispiel Dateien nicht öffnen lassen, weil ein Teil der Pfadangabe nicht stimmt oder sie nicht beschrieben werden können, weil der Benutzer nicht über die notwendigen Rechte verfügt oder ein Verzeichnis nicht mehr existiert, in das eine Datei kopiert werden soll. Es kann aber auch nicht bedacht sein, dass eine Methode einen absoluten (System-)Pfad benötigt und dann ein bestimmter Fehler ausgelöst wird. In allen Fällen wird eine Fehler-Erkennung und Fehler-Behandlung erforderlich.

Wie in Gambas üblich werden vorwiegend die folgenden Schlüsselwörter TRY, FINALLY und CATCH, das Konstrukt "IF ERROR THEN ... ENDIF" sowie das Ereignis Application_Error() eingesetzt:

- Mit 'TRY Anweisung' wird versucht eine Anweisung auszuführen, die im Fehlerfall keine Fehler-Anzeige auslöst! Das mag sinnvoll erscheinen, doch ist zu bedenken, dass der Fehler so nur kaschiert wird!
- Das Konstrukt "IF ERROR THEN … ENDIF" wird daher unmittelbar nach 'TRY Anweisung' verwendet, um zu erfahren, ob die Anweisung korrekt ausgeführt wurde oder nicht.
- Ein Finally-Block wird am Ende einer Funktion oder einer Prozedur immer ausgeführt – ob während der Ausführung davor ein Fehler aufgetreten ist oder nicht. Wenn es in der Methode zusätzlich einen Catch-Block gibt, muss ein Finally-Block vor diesem stehen.
- Ein Catch-Block steht am Ende am Ende einer Funktion oder einer Prozedur oder nach einem Finally-Block, eingeleitet durch das CATCH-Schlüsselwort. Der Catch-Block wird nur dann ausgeführt, wenn zwischen dem Beginn der Ausführung einer Methode und ihrem Ende ein Fehler aufgetreten ist. Wenn während der Ausführung des Catch-Blocks ein Fehler auftritt, wird dieser angezeigt.
- Ein ausgelöster Fehler wandert sofort den Aufruf-Stack im Interpreter hinauf – solange bis der Fehler beachtet oder behandelt wird. Das kann ein TRY vor einer Anweisung sein, das den Fehler überdeckt im Zusammenhang mit "IF ERROR THEN …" oder ein Finally, das auch einen Fehler annimmt und noch Daten retten will oder ein CATCH, das den Fehler verdeckt behandelt. Findet der Interpreter weder das Schlüsselwort Try oder einen Finally- oder Catch-Block oder das Event Application_Error(), dann stürzt der Interpreter ab!
- Zusätzlich kann man innerhalb einer Fehlerbehandlung auch auf die angezeigten Fehlernummern (http://gambaswiki.org/wiki/error [1]) abzielen, um zum Beispiel aussagekräftige Fehlermeldungen in deutscher Sprache zu erzeugen.
- Eine Besonderheit stellt in diesem Zusammenhang der Einsatz des Ereignisses Application_Error() dar. Dieses Ereignis wird ausgelöst, wenn ein Fehler auftritt, der von keiner Try-Anweisung herrührt oder von einem Finally- oder Catch-Block behandelt wird.
- Es gibt aber auch präventive Strategien, bei der bestimmte Fehler(-Klassen) ausgeschlossen werden können.

11.5.1.1 TRY

```
[1] Try TextArea1.Text = File.Load(".../help/help.txt")
[2] If Error Then
[3]    Message.Error("The help file cannot be loaded.")
[4]    FHelp.Delete()
[5] Endif
```

Tritt beim Versuch, die Hilfe-Datei zu laden, *kein* Fehler auf, wird deren Inhalt in einer Text-Area angezeigt. Kann die Hilfe-Datei aber nicht geladen werden, dann wird der ausgelöste Fehler nicht angezeigt – er wird versteckt. Deshalb werden in diesem Fall die Zeilen 2 bis 5 ausgeführt, die zumindest über einen vorliegenden Fehler informieren. Im vorliegenden Fall muss auf das Anzeigen der Hilfe-Datei verzichtet werden – das Programm aber läuft weiter.

11.5.1.2 FINALLY

Im nächsten Beispiel folgt dem Finally-Block ein Catch-Block:

```
[1] Private Sub AddTextToFile(FilePath As String, Text As String)
[2]
[3]   Dim hFile As File
[4]
[5]   hFile = Open FilePath For Append
```

```
[6]   Finally
[7]     If Exist(FilePath) Then Close #hFile
[8]   Catch
[9]     Message.Error("Error:\n" & Error.Text & " in " & Error.Where)
[10] End
```

In jedem Fall wird die Datei geschlossen – dafür sorgt der Finally-Block. Der (optionale) Catch-Block würde einen auftretenden Fehler in den Zeilen 5 bis 7 anzeigen.

11.5.1.3 CATCH

Das folgende Beispiel nutzt nur einen Catch-Block, um hinreichend differenziert alle auftretenden Fehler mit einer passenden Fehlermeldung anzuzeigen:

```
[0] Public Sub btnHelpLibrary_Click()
[1]   Dim cl As Class
[2]
[3]   cC = Component.Load(":gambasbook/LPathProject:1.2")
[4]   cl = Class.Load("Module1")
[5]   Object.Call(cl, cl.Symbols[0], Null)
[6]   Catch
[7]     Message.Error(Error.Text)
[8]
[9] End
```

Die beiden u.a. Fehlermeldungen werden verständlich, wenn einerseits die Bibliothek LPathProject nicht geladen werden kann (Fehler 1) und damit andererseits auch das Module1 nicht verfügbar ist (Fehler 2). Dagegen gab es nur die zweite Fehlermeldung, als in der bestehenden Bibliothek das Modul mit dem angegebenen Namen `Module1` nicht existierte.

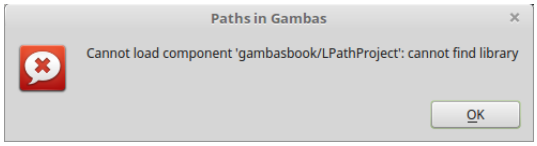

Abbildung 11.5.1.3.1: Erste Fehlermeldung

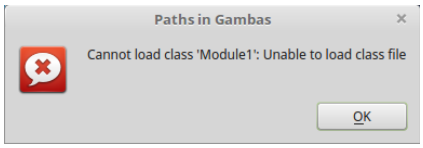

Abbildung 11.5.1.3.2: Zweite Fehlermeldung

Der Catch-Block im folgenden Beispiel fängt konsequent alle ausgelösten Fehler ab und gibt eine detaillierte Fehlermeldung aus. Zusätzlich werden in den letzten beiden Prozeduren die geöffneten Dateien geschlossen – sofern die Datei-Pfade existieren:

```
Public Sub btnAddTextToFile_Click()

  Dim sLogDir As String

  sLogDir = Desktop.DataDir &/ "gambasbook" &/ AppName
  If Not Exist(sLogDir) Then Shell.MkDir(sLogDir)
  AddTextToFile(sLogDir &/ "rs232.log", "Time = " & Format(Now, "hh:nn:ss") & " - " & sTValue)
  Catch
    Message.Error("Error:\n" & Error.Text & " in " & Error.Where)
End

Public Sub btnGetTextFromFile_Click()

  Dim sLogPath As String

  sLogPath = Desktop.DataDir &/ "gambasbook" &/ AppName &/ "rs232.log"
  txaLog.Text = GetTextFromFile(sLogPath)
  Catch
    Message.Error("Error:\n" & Error.Text & " in " & Error.Where)
End

Private Sub AddTextToFile(FilePath As String, Text As String)

  Dim hFile As File
```

```
  hFile = Open FilePath For Append
  Finally
    If Exist(FilePath) Then Close #hFile
  Catch
    Message.Error("Error:\n" & Error.Text & " in " & Error.Where)
End

Private Function GetTextFromFile(FilePath As String) As String

  Dim hFile As File
  Dim sLine, Text As String

  hFile = Open FilePath For Read

  While Not Eof(hFile)
    Line Input #hFile, sLine
    Text &= sLine & "\n"
  Wend

  Return Text

  Finally
    If Exist(FilePath) Then Close #hFile
  Catch
    Message.Error("Error:\n" & Error.Text & " in " & Error.Where)

End
```

11.5.1.4 Ereignis Application_Error()

Annahme:
Ein Fehler u.U. wird ausgelöst, für den weder eine Try-Anweisung noch ein Finally- oder Catch-Block vorgesehen ist. Genau dann werden Sie zum Einsatz des Ereignisses *Application_Error()* greifen, um zum Beispiel Daten unbedingt noch in eine Datei zu schreiben oder um eine Datei unter allen Umständen zu löschen, bevor das Programm beendet wird. Dieses Ereignis wird in der Startklasse des Projekts ausgelöst. Nur dort kann die Ereignisbehandlungsroutine stehen, damit etwas passiert und sie muss die Signatur 'Static Public Sub Application_Error()' haben. Die Startklasse darf kein Modul sein. Im Beispiel wird eine Sperr-Datei angelegt, die jeden Versuch, eine weitere Instanz der Anwendung anzulegen unterbinden soll. Wird das Programm fehlerfrei beendet, so wird die Sperr-Datei gelöscht. Wird die Prozedur RunATest() ausgeführt, dann erfolgt auch die Division durch den Divisor Null. Ein Fehler wird ausgelöst, der weder von einer Try-Anweisung noch von einem Finally- oder Catch-Block behandelt wird. Das Programm würde abstürzen – und damit bleibt die Sperr-Datei *lock.file* im System zurück. Das hätte zur Folge, dass jeder weitere Start des Programms unterbunden wird, weil es bereits eine Sperr-Datei gibt!

Einen Ausweg bietet das Ereignis *Application_Error()*, in dessen Ereignisbehandlungsroutine die Sperrdatei *lock.file* gelöscht wird – bevor alle Lichter ausgehen … .

```
[1]  ' Gambas class file
[2]
[3]  Public bFlag As Boolean = False
[4]
[5]  Public Sub _new()
[6]
[7]    If Exist(Application.Path &/ ".lock.file") Then
[8]       Message.Warning(Subst("&1 '&2' &3", ("There is already an instance of"), Application.Name, "!"))
[9]       bFlag = False
[10]      FMain.Close()
[11]   Endif
[12]
[13]   Shell "touch " & Application.Path &/ ".lock.file" Wait
[14]
[15]   bFlag = True
[16]
[17] End
[18]
[19] Static Public Sub Application_Error()
[20]   If Exist(Application.Path &/ ".lock.file") Then Kill Application.Path &/ ".lock.file"
[21]   Wait
[22] End
[23]
[24] Public Sub Form_Open()
[25]   FMain.Resizable = False
[26] End
[27]
```

```
[28] Public Sub Form_Close()
[29]     If bFlag Then Try Kill Application.Path &/ ".lock.file"
[30]     FMain.Close()
[31] End
[32]
[33] Public Sub btnTest_Click()
[34]  RunATest()
[35] End
[36]
[37] Private Sub RunATest()
[38]
[39]   Dim i As Integer
[40]   Dim q As Float
[41]
[42]   For i = -2 To 2
[43]     q = 5 / i
[44]   Next
[45]
[46] End
```

Abbildung 11.5.1.4.1: GUI

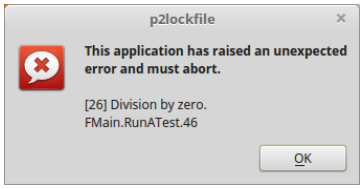

Abbildung 11.5.1.4.2: Test-Fehlermeldung

Fazit:

- Mit der Verwendung von `TRY` und `IF ERROR THEN ... ENDIF` können Sie nicht nur einen Fehler erkennen, sondern auch in vielfältiger Weise behandeln.
- Mit `FINALLY` wird Ihnen die Gelegenheit gegeben, eine Anweisung immer ausführen zu lassen; auch im Fehlerfall.
- Der Einsatz von `CATCH` ist genau abzuwägen, da ein aufgetretener Fehler nur angezeigt wird, aber nicht *unmittelbar* behandelt wird.
- Der Einsatz des Ereignisses *Application_Error()* ist nur in speziellen Fällen sinnvoll.