### 11.5.0.3 Fehlerbehandlung 4

In diesem Kapitel stehen das Debugging in der IDE, das Profiling, das Logging und die Beschreibung der Eigenschaft *System.BreakOnError* im Mittelpunkt.

#### 11.5.0.3.1 Debugging in Gambas

Wenn Sie ein Modul oder eine Klasse sowie die darin enthaltenen Prozeduren in Ihrem Gambas-Projekt testen wollen, dann stellt Ihnen die IDE geeignete Werkzeuge zur Verfügung. Als Entwickler benötigen Sie für effektives Arbeiten:

- den verwendeten Algorithmus der zu testenden Prozedur,
- ein geeignetes Daten-Variablen-Konzept,
- ein Fehler-Management, das Sie mit Try, Catch, Finally, dem Ereignis Application_Error(), der booleschen Variable Error, der Klasse Error und den Anweisungen Print, Debug und Error im Quelltext umsetzen.
- Geeignete Daten zum Testen.

Diese folgende Vorgehensweise hat sich bewährt:

- Entscheiden Sie zuerst, was Sie an Programm-Ausgaben oder aktuellen Werten von Steuer-Elementen interessiert:

Button 'Konsole'

In der Konsole der IDE werden alle Ausgaben der Anweisungen Print, Debug, Error und allgemeinen (System-)Fehler-Meldungen ausgegeben wie diese zum Beispiel:

```
gb.gui.qt: warning: 'gb.qt5' component not found, using 'gb.qt4' instead
"sni-qt/13396" WARN  17:23:08.002 void StatusNotifierItemFactory::connectToSnw() Invalid interface to ...
```

Button 'Fehlersuche' zur Programm-Laufzeit

- Liste 'Aktuelles Objekt'.
  Sie enthält Steuerelemente und Werte der globalen Variablen, die fett markiert sind.
- Liste 'Lokale Variablen'.
  Die Liste kann auch leer sein, da die Ausgaben von den in der aktuellen Prozedur deklarierten lokalen Variablen abhängen.

Die Listen müssen Sie bei Bedarf ausklappen.

- Legen Sie Haltepunkte (Breakpoints) im Quelltext mit der Funktionstaste F9 fest, wenn das notwendig ist. Mit F9 kann ein Haltepunkt auch wieder gelöscht werden.
- Starten Sie das Programm im Einzelschritt-Modus mit F8.
- Wechseln Sie zwischen F8 und den Aktionen in einem Fenster mit Tastatur und Maus, um das Programm im Einzelschritt laufen zu lassen.
- Legen Sie fest, ob Sie einen Ausdruck überwachen wollen: Markieren Sie den Ausdruck im Quelltext und wählen dann mit der rechten Maustaste im Kontext-Menü den Eintrag 'Ausdruck überwachen'. Die Liste 'Ausdrücke überwachen' wird neu angelegt oder erweitert.
- Sehen Sie sich die u.U. geänderten Werte in den Listen 'Aktuelles Objekt' und 'Lokale Variablen' an. Bei einfachen Daten-Typen sehen Sie den Wert direkt im Fenster 'Fehlersuche'. Bei komplexen Datentypen wie einem Array oder einem Steuer-Element - erkennbar an den runden Klammern - können Sie die erste Ebene der relevanten Werte mit einem Doppelklick erreichen. Hier sehen Sie die Werte der einfachen Daten-Typen und kommen auf die nächste Ebene bei komplexen Datentypen wieder mit einem Doppelklick.


Abbildung 11.5.0.3.1: Fenster 'Fehlersuche'

In Abhängigkeit von den Ergebnissen werden Sie den Quelltext punktuell ändern sowie erneut testen und kommen so schrittweise dem Ziel näher, ein korrekt arbeitendes Programm zu entwickeln. Zugegeben, das folgende Beispiel ist konstruiert – es zeigt aber die Arbeitsweise, um einem angezeigten Fehler auf die Spur zu kommen. Das ist der Quelltext-Abschnitt:

```
Public Sub Form_Open()
  Dim x,y As Integer
  Dim a As Float
  Do
    x = Rnd(-3,3)
    y = Rnd(-3,3)
    a = 1/x + 2/y
  Loop
End
```

Irgendwann zeigt sich dieser Fehler:

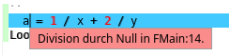


Abbildung 11.5.0.3.2: Fehler

Jetzt gilt es zu erkunden, was den Fehler ausgelöst hat – x oder y? Klicken Sie in der fehlerhaften Zeile doppelt auf das x oder markieren Sie x – dann wird 2 angezeigt:

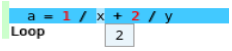


Abbildung 11.5.0.3.3: Wert von x

X war es also nicht. Mit einem Doppelklick auf y zeigt sich das:

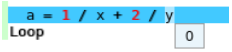


Abbildung 11.5.0.3.4: Fehler erkannt ...

Der zweite Quotient 2/y ist nicht definiert: Division durch Null!

### 11.5.0.3.2 Logging

Im Kapitel 19.5 Logging wird die Komponente gb.logging beschrieben, die ein flexibles System für die Protokollierung in Gambas-Anwendungen implementiert. 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).

### 11.5.0.3.3 Profilierung

Im Gambas ist Profiling eine Form der dynamischen Programmanalyse, die den Speicher, die zeitliche Komplexität, die Verwendung bestimmter Anweisungen oder die Häufigkeit und Dauer von Funktionsaufrufen misst.

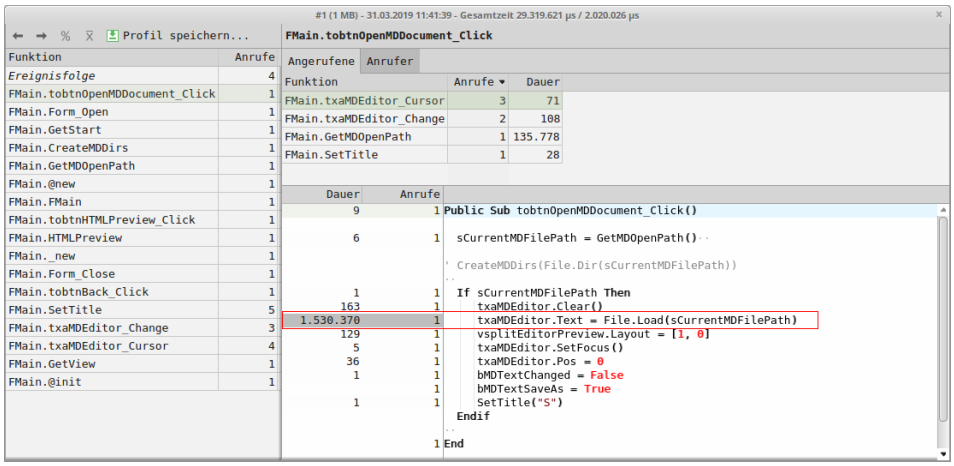


Abbildung 11.5.0.3.5: Ergebnisse bei aktiviertem Profiling

Die angezeigten Informationen dienen zur Unterstützung bei der Fehlersuche in zeit-kritischer Umgebung und zur Programmoptimierung.

Sie können den Profiling-Modus in der IDE im Menü `Debuggen> Profilierung aktivieren` einschalten. Im Quelltext können Sie über die boolesche Variable *System.Profile* abfragen, ob der Profiling-Modus eingeschaltet ist oder nicht. Die statische boolesche Eigenschaft System.Profile (gb) gibt den Wert True zurück, wenn die Profil-Erzeugung aktiviert ist. Die Eigenschaft können Sie auch setzen.

Mit `System.Profile = True` und `System.Profile = False` können Sie das Profiling ein- und ausschalten. Das kann notwendig werden, wenn Sie nur einen bestimmten Quelltext-Abschnitt beobachten wollen. Außerdem kann das Profiling langsam sein und es gibt eine Begrenzung der Größe der Profildatei. Die Standardgröße beträgt 512 MiB. Setzen Sie die Umgebungsvariable `GB_PROFILE_MAX` auf eine positive Ganzzahl (Einheit in 'MiB'), dann können Sie die Größe von von 128 MiB bis 4 GiB setzen. Achtung: Wird das Programm in der IDE nicht mit aktiviertem Profiling-Modus ausgeführt, so wird die Eigenschaft *System.Profile* ignoriert.

#### 11.5.0.3.4 System.BreakOnError

Gambas bietet mit der Eigenschaft *System.BreakOnError* eine sehr spezifische Eigenschaft. Diese Eigenschaft erlaubt die Implementierung des Features "Break on error" in der IDE. Die statische Eigenschaft System.BreakOnError gibt an, ob ein Programm in der IDE anhält, wenn ein Fehler ausgelöst wird – auch wenn der Fehler ordnungsgemäß im Programm abgefangen wird. Die Eigenschaft kann auch gesetzt werden.  Wenn Sie diese Eigenschaft auf True setzen, dann können Sie keine Fehler mehr abfangen. Jeder Fehler führt sofort zum Abbruch des Programms – unabhängig davon, ob Sie den Fehler mit Try oder Catch abfangen oder nicht.