21.3.1 Einsatz Quick-Syntax

Die einfachste Art Daten aus einem Prozess zu *lesen*, besteht darin, die Kurzform der SHELL- oder EXEC-Instruktion – die sogenannte Quick-Syntax – zu verwenden:

```
SHELL sCommand TO (String-)Variable       ' sCommand ist ein String
EXEC  aCommand TO (String-)Variable       ' aCommand ist ein Array
```

Wenn Sie die Quick-Syntax verwenden, wird der angegebene Befehl *Command* ausgeführt und der Interpreter *wartet* auf dessen Ende! Danach wird die komplette Ausgabe der Daten in die angegebene (String-)Variable geschrieben – sofern kein Prozess-Fehler (!) auftrat.

Beispiel 1:

```
Public Sub btnExecuteInstruktion_Click()
  EXEC ["pstree", "-p"] TO TextArea.Text
End
```

Die vollständige Ausgabe des Prozessbaumes wird in einer TextArea angezeigt. Schneller und eleganter geht es kaum.

Wenn der Prozess nicht gestartet werden konnte – statt *pstree* steht zum Beispiel nur *pstre* – dann erhalten Sie in der IDE die Fehlernachricht "Cannot run child process: cannot exec program …". Einen solchen (Start-)Fehler können Sie im Programm so abfangen:

```
Public Sub btnExecuteInstruktion_Click()
  TextArea1.Clear
  Try Exec ["pstree", "-p"] TO TextArea1.Text
  If Error Then
     Message.Error("Fehler beim Ausführen des Befehls!")
     Return
  Endif ' ERROR ?
End ' ExecuteInstruktion_Click()
```

Beispiel 2:

Es soll aus einem Gambas-Programm heraus nur geprüft werden, ob das Kompilieren eines bestimmten Gambas-Projektes erfolgreich war oder ein Fehler beim Kompilieren auftrat:

```
Public Sub btnExecuteInstruktion_Click()
  Dim sOutput As String

  Shell "gbc3 $HOME/color_select 2>/dev/null || echo $?" TO sOutput

  If Upper(Left(sOutput, 2)) <> "OK" Then
     Message.Error("Beim Kompilieren trat ein Fehler auf!")
  Else
     Message.Info("Das Kompilieren war erfolgreich!")
  Endif ' Upper(Left(sOutput, 2)) <> "OK" ?

End ' ExecuteInstruktion_Click()
```

Als Ausgaben können Sie bei erfolgreichem Kompilieren das 'OK' aus der Standard-Ausgabe oder bei einem Fehler des Programms 'gbc3' über die Bash-Variable $? einen Rückgabewert ≥ 1 erwarten. Die Standard-Fehlerausgabe wurde in diesem Beispiel ins elektronische Nirwana umgelenkt. Wollen Sie nur einen Fehler abfangen und anzeigen, verzichten Sie auf den Else-Zweig.

Beispiel 3 – Projekt

Das Beispiel 3 stellt ein vollständiges Projekt zur Quick-Syntax vor. Das externe Programm ist 'ping', mit dem Sie für einen bestimmten Server die Signallaufzeiten messen. Unter 'man ping' oder 'info ping' oder 'ping –help' finden Sie interessante Details zum Konsolen-Programm *ping*. Die URL ist frei wählbar. Die Anzahl der Pings ist auf 4 festgesetzt worden und kann nicht geändert werden. Im Gambas-Programm ist die Quick-Syntax für SHELL und EXEC verwendet worden. Eine LED signalisiert den Programmzustand respektive den des gestarteten Prozesses, in den Sie nicht eingreifen können!

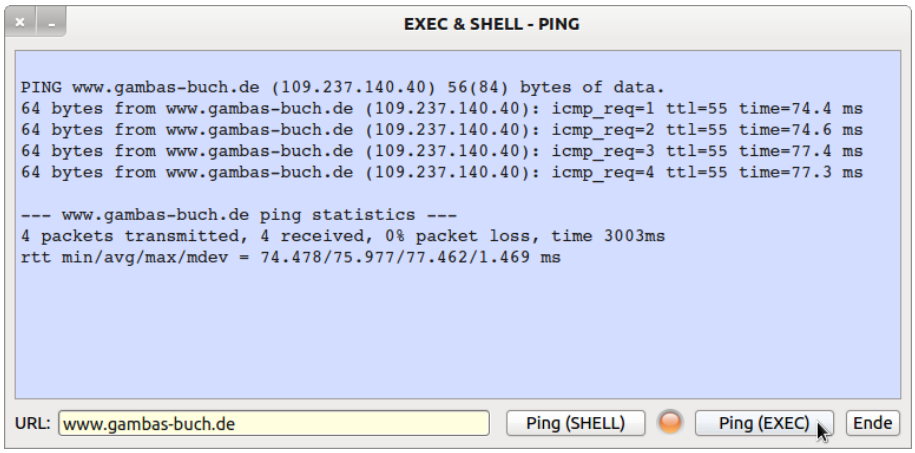

Abbildung 21.3.1.1: GUI für das Programm 'ping'

Quelltext:

```
' Gambas class file

Private sProgrammName As String = "ping"

Public Sub Form_Open()
  FMain.Center
  FMain.Resizable = False
  SetLEDColor("orange")
End ' Form_Open()

Public Sub btnPingOverShell_Click()
  Dim sAusgabe, sCommand As String

  SetLEDColor("green")
  TextArea.Clear
  Wait
  FMain.Mouse = Mouse.Wait

  sCommand = sProgrammName & Chr(32) & TextBox1.Text & " -c 4"
  Shell sCommand To TextArea.Text

  FMain.Mouse = Mouse.Default
  SetLEDColor("orange")
End ' PingOverShell

Public Sub btnPingOverExec_Click()
 Dim sAusgabe As String
 Dim aCommand As New String[]

  SetLEDColor("green")
  TextArea.Clear
  Wait
  FMain.Mouse = Mouse.Wait

  aCommand = [sProgrammName, TextBox1.Text, "-c", "4"] ' Inline-Array
  Exec aCommand To sAusgabe

  TextArea.Insert(gb.NewLine & sAusgabe)
  FMain.Mouse = Mouse.Default
  SetLEDColor("orange")
End ' PingOverExec

Public Sub SetLEDColor(sLEDColor As String)
  PictureBox1.Picture = Picture["LED/led_" & sLEDColor & ".svg"]
End ' SetLEDColor(..)

Public Sub btnClose_Click()
  FMain.Close
End ' btnClose_Click()
```

Kommentare:

- Die komplette Ausgabe in eine TextArea ist verlockend, es wird aber der Inhalt der TextArea überschrieben. Sie haben keine Möglichkeiten, die Ausgabe zu analysieren und aufzubereiten.
- In der zweiten Prozedur wird nur angedeutet, wie die Ausgabe des Prozesses in einer Variable gespeichert wird und die Ausgabe aufbereitet wird, indem eine Leerzeile am Anfang eingefügt wird.

Im folgenden Exempel dagegen wird die Ausgabe umfangreich bearbeitet, um den Inhalt für eine ComboBox im Hauptprogramm bereitzustellen:

```
PUBLIC SUB RS232ListeGenerieren()
  DIM iCount AS Integer
  DIM sZeile, sListeV24, sListeUSB, s AS String
  DIM aSchnittstellenMatrix AS NEW String[]
  DIM aListe AS NEW String[]

  cmbRS232PortName.Clear() ' ComboBox-Inhalt löschen

' Ermittlung echter RS232-Schnittstellen
  SHELL "dmesg | grep ttyS | grep 00:" TO sListeV24
  IF Len(sListeV24) > 0 THEN
     aSchnittstellenMatrix = Split(sListeV24, " ")
     FOR EACH sZeile IN aSchnittstellenMatrix
         IF InStr(sZeile, "ttyS") THEN
            cmbRS232PortName.Add("/dev/" & Trim$(sZeile))
         ENDIF
     NEXT ' FOR EACH sZeile
  ENDIF ' Len(sListeV24) > 0 ?

' Ermittlung USB-RS232-Adapter-Schnittstellen
  SHELL "dmesg | grep ttyUSB" TO sListeUSB
  IF Len(sListeUSB) > 0 THEN
     aSchnittstellenMatrix = Split(sListeUSB, "\n")
     FOR EACH sZeile IN aSchnittstellenMatrix
         FOR iCount = 0 TO 7
             IF InStr(sZeile, "ttyUSB" & CInt(iCount)) THEN
                aListe.Add("ttyUSB" & CInt(iCount))
             ENDIF
         NEXT ' iCount
     NEXT ' FOR EACH sZeile
  ENDIF ' Len(sListeUSB) > 0 ?

  aListe.Sort
  aListe = RemoveMultiple(aListe)

  FOR iCount = 0 TO aListe.Max
      PRINT aListe[iCount]
      cmbRS232PortName.Add("/dev/" & Trim$(aListe[iCount]))
  NEXT ' iCount

  IF cmbRS232PortName.Count = 0
     cmbRS232PortName.Background = Color.RGB(255, 191, 191)
     cmbRS232PortName.Add("Keine RS232-Schnittstelle gefunden!")
  ENDIF
END ' RS232SchnittstellenlisteGenerieren

PUBLIC FUNCTION RemoveMultiple(aStringListe AS String[]) AS String[]

  DIM iCount AS Integer
  DIM iIndex AS Integer
  DIM sElement AS String

  iIndex = 0
  WHILE iIndex < aStringListe.Count
  iCount = 0
    sElement = aStringListe[iIndex]
    WHILE aStringListe.Find(sElement) <> -1
      INC iCount
      aStringListe.Remove(aStringListe.Find(sElement))
    WEND
```

```
    IF iCount MOD 2 = 1 THEN
       aStringListe.Add(sElement, iIndex)
       INC iIndex
    ENDIF
  WEND

  RETURN aStringListe

END ' RemoveMultiple(..)
```