21.2 Instruktionen SHELL und EXEC

In den folgenden Abschnitten werden die Syntax für die beiden Instruktionen SHELL und EXEC vorgestellt und erläutert sowie vergleichende Betrachtungen zu diesen Instruktionen angestellt.

21.2.1 Syntax SHELL

Es gibt zwei Varianten für die SHELL-Instruktion – eine Version mit vielen unterschiedlichen Optionen und eine einfache Variante, um einen Prozess zu starten:

```
[ ProzessVariable = ] SHELL Command [WAIT] [FOR {{READ|INPUT}|{WRITE|OUTPUT}}] [AS PEventName]
```

Hinweise:

- Sie müssen nicht notwendigerweise eine Prozess-Variable deklarieren. Nur wenn Sie WRITE oder OUTPUT verwenden ist das zwingend erforderlich.
- SHELL ist der Name der verwendeten Instruktion.
- *Command* ist eine Zeichenkette und beschreibt den Shell-Befehl mit Programm-Name, Optionen, Parametern und Umlenkungen sowie Pipes.
- Es wird der Shell-Befehl (Command) in einer System-Shell ausgeführt und ein Prozess-Objekt erzeugt.
- Verwenden Sie *WAIT*, dann wartet der Interpreter auf das Prozessende.
- *FOR READ* zeigt an, dass der Prozess zum Lesen geöffnet werden soll. Die Standard-Ausgabe des Prozesses wird auf das laufende Gambas-Programm umgeleitet. Sie können alternativ auch *FOR INPUT* verwenden.
- Verwenden Sie *FOR WRITE* oder *FOR OUTPUT , w*enn Sie Eingaben des Benutzers an den Prozess übergeben wollen! Die Standard-Eingabe des Prozesses wird auf das Programm umgeleitet.
- Sie können auch *FOR READ WRITE* oder *FOR INPUT OUTPUT* verwenden, um die Ausgaben des Shell-Befehls zu lesen und Eingaben des Benutzers an den Prozess zu übergeben.
- *FOR INPUT OUTPUT* lässt das interne Stream-Objekt zeilenorientiert arbeiten. Alles wird in Einheiten aus "Zeilen" abgearbeitet wie es in einer Shell üblich ist.
- Dem *FOR READ* oder *FOR INPUT* oder *FOR READ WRITE* oder *FOR INPUT OUTPUT* muss ab Gambas 3 ein *'AS PEventName'* folgen. *PEventName* ist vom Typ String.
- Die Angabe *'AS PEventName'* deklariert den Prozess – der mit SHELL erzeugt wurde – als einen zu überwachenden Prozess und stellt ein Symbol für den Event-Überwacher von Gambas bereit. Der Event-Überwacher ruft Funktionen des Programms auf, wenn ein spezielles Prozess-Ereignis (Event) eintritt. Die ausgelöste Funktion wird als Event-Handler bezeichnet. Wenn zum Beispiel Daten auf der umgeleiteten Standard-Ausgabe des Prozesses verfügbar sind, löst der Event-Überwacher den Event-Handler *PEventName_Read()* aus, damit dieses Ereignis behandelt werden kann. Es könnte darin bestehen, die eingelesenen Daten zu verarbeiten und die relevanten Daten formatiert auszugeben oder in Abhängigkeit von den Daten den Programmablauf zu ändern. Es gibt für den Prozess zwei weitere Event-Handler: *PEventName_Kill() und PEventName_Errror(..).* Diese werden ausgelöst, wenn der Prozess beendet wurde oder Daten an der umgeleiteten Standard-Fehlerausgabe des Prozesses vorliegen.

Folgende Aufrufe sind korrekt:

```
SHELL "firefox www.gambas-buch.de"
SHELL "dmesg | grep ttyS | grep 00: > /tmp/schnittstellenliste.liste" WAIT
SHELL "dmesg | grep ttyUSB >> /tmp/schnittstellenliste.liste" WAIT

Private $hProcess As Process

aCommand = ["ping", "www.fewo-kellermann.de", "-c", "4"]
$hProcess = EXEC aCommand For Read As "myPingProcess"

aCommand = ["wget", "www.gambas-buch.de", "-O-", "--directory-prefix=" & User.Home]
$hProcess = EXEC aCommand For Read As "myProcess"

sCommand = "mysql -f -n -vvv -u root -pYourMySQLPassword4MySQLRoot"
$hProcess = SHELL sCommand For Input Output As "myProcess"
```

Die 2. Syntax (Quick-Syntax) ist einfach:

```
SHELL Command TO (String-)Variable
```

Die Ausgaben des im Hintergrund gestarteten Prozesses werden in der (String-)Variable gespeichert.

Achtung: Sie haben keine Kontrolle über den *blockierend* ausgeführten Prozess! Wenn der Prozess nicht gestartet werden konnte, dann können Sie diesen Fehler abfangen.

21.2.2 Syntax EXEC

Es gibt auch zwei Varianten für die EXEC-Instruktion, um einen Prozess zu starten:

```
[ ProzessVariable = ] EXEC Command [WAIT] [FOR {{READ|INPUT}|{WRITE|OUTPUT}}] [AS PEventName]
```

- Es werden nur die Änderungen gegenüber der SHELL-Instruktion beschrieben.
- *Command* ist ein Array. Das erste Element im Array, das auch ein Inline-Array sein kann, ist der Name des Befehls und die anderen Elemente sind die Befehlsparameter – sofern welche übergeben wurden.
- Es wird der Exec-Befehl (Command) als execve()-System-Aufruf direkt ausgeführt und ein Prozess-Objekt erzeugt, mit dessen Hilfe man die Ausgaben des Befehls sowie notwendige Eingaben des Benutzers steuern kann oder Prozess-Fehler erkennt.

Folgende Aufrufe sind korrekt:

```
Private $hCommand As Process
$hCommand = Exec [...] For Read Write As "MyCommand"
```

Zweite Variante der EXEC-Instruktion:

```
EXEC Command TO (String-)Variable
```

Wenn Sie die zweite Syntax verwenden, wird der angegebene Exec-Befehl ausgeführt und der Interpreter *wartet* auf dessen Ende. Danach wird die komplette Ausgabe des im Hintergrund gestarteten Prozesses in die angegebene String-Variable geschrieben. Auch in diesem Fall gilt mit allen Konsequenzen: Sie haben keine Kontrolle über den ausgeführten Prozess!

21.2.3 Syntax-Erweiterung – Umgebungsvariablen

Sie können für den zu startenden Prozess ausgewählte Umgebungsvariablen auf neue Werte setzen, indem Sie nach dem Shell- oder Exec-Befehl *Cmd* das Schlüsselwort WITH angeben, dem mit *Environment* ein Array aus Umgebungsvariablen-Werte-Paaren folgt:

```
[Process=] EXEC Cmd WITH Environment …
[Process=] EXEC Cmd WITH Env_Array [WAIT] [FOR {{READ|INPUT}|{WRITE|OUTPUT}}] [AS PEventName]
```

```
[Process=] SHELL Cmd WITH Environment …
[Process=] SHELL Cmd WITH Env_Array [WAIT] [FOR {{READ|INPUT}|{WRITE|OUTPUT}}] [AS PEventName]
```

Diesen Aufruf in einer Konsole :

```
hans@linux:~$ LC_ALL=en_GB.utf8 gambas3
```

können Sie in einem Gambas-Programm nachbilden, um mit der SHELL-Instruktion oder der EXEC-Instruktion zum Beispiel Gambas 3 temporär in der englischen Version zu starten:

```
Public Sub btnStartGambas3EN_Click()
 SHELL "gambas3" With ["LC_ALL=en_GB.utf8"] Wait
 EXEC ["gambas3"] With ["LC_ALL=de_DE.utf8"] Wait ' Alternative: EXEC-Instruktion
End
```

21.2.4 Vergleichende Betrachtungen zu den Instruktionen SHELL und EXEC

Ob Sie die SHELL-Instruktion oder die EXEC-Instruktion verwenden, wird vor allem durch die zu bearbeitende Aufgabe bestimmt. Zuerst werden Unterschiede zwischen den Instruktionen SHELL und

EXEC beschrieben und dann Gemeinsamkeiten herausgestellt. Damit können Sie sicherer entscheiden, ob Sie die SHELL-Instruktion oder die EXEC-Instruktion einsetzen, um erfolgreich zu arbeiten.

21.2.4.1 Unterschiede SHELL – EXEC

Es besteht ein wesentlicher Unterschied in der Art und Weise wie der übergebene Befehl (Command) ausgeführt wird und im Typ des Befehls:

- SHELL benutzt zum Beispiel die Shell /bin/sh und EXEC direkt den Systemaufruf *execve*, was auch der Grund dafür ist, dass EXEC seine Argumente als *Array* benötigt und SHELL als *String*. Für Details zum Systemaufruf sehen Sie bitte in den Manpages unter *$ man 2 execve* nach.
- Der Systemaufruf *execve* übernimmt den Programmpfad, das Argumente- und das Umgebungsvariablen-Array und startet das Programm dementsprechend. Auf diese Weise kann aber nur ein externes Programm gestartet werden.
- Konstrukte wie: "prog1; prog2" oder "prg1 | prg2" oder "programm 2>datei" sind Funktionalitäten einer Shell und stehen bei der Verwendung von EXEC nicht zur Verfügung.
- EXEC ist jedoch schneller, da die SHELL-Instruktion erst eine Shell starten muss, dann die (optional) übergebenen Parameter auswertet und erst danach das (externe) Programm startet.
- Anders als SHELL ist EXEC eine Schnittstelle zum Kernel. Diese bietet weniger Möglichkeiten als die Nutzung einer Shell, ist allerdings wesentlich sicherer – und externe Programme starten kann man damit in jedem Fall.

Für den Einsatz der SHELL-Instruktion sprechen diese Vorzüge, die in einer Shell verfügbar sind:

- Ersetzungen über Wildcards wie '*.txt' .
- Umleitungen und Pipes wie '> ~/output.txt' oder '< ~/input.dat' oder ' 2>&1 | Kommando' o.ä..
- Verwendung von speziellen Variablen wie etwa $HOME oder $PATH oder $UID oder $#.

Quoting

Ab Gambas 3 kann man mit der Funktion *Shell$(Shell-Befehl-String)* eine sichere Übergabe an die Shell realisieren, damit bestimmte Zeichen im Shell-Befehl-String von der verwendeten Shell – bei Ubuntu ist das '/bin/dash' – nicht interpretiert werden. Für EXEC ist ein Maskieren ausgewählter Zeichen nicht erforderlich. Unter der folgenden URL finden Sie gut aufbereitete Informationen zum Quoting: *http://wiki.ubuntuusers.de/Shell/Bash-Skripting-Guide_für_Anfänger* .

21.2.4.2 Gemeinsamkeiten SHELL – EXEC

Die SHELL-Instruktion und die EXEC-Instruktion starten beide ein externes Programm – ein Prozess wird erzeugt. Wesentliche Eigenschaften, Methoden und Events sowie ausgewählte Konstanten der Klasse '*Process*' sind im Kapitel 21.1 beschrieben worden. Im folgenden Kapitel 21.3 werden zuerst der Einsatz der Event-Handler *PEventName_Read()*, *PEventName_Kill() und PEventName_Error(..)* beschrieben. Um Eingaben an den Prozess übergeben, werden dafür entsprechende Prozeduren vorgestellt. Diese Beschreibungen gelten für die beiden Instruktionen SHELL und EXEC in gleicher Weise.