10.3.2 FOR-EACH-Kontroll-Struktur

In diesem Kapitel wird die FOR-EACH-Kontroll-Struktur beschrieben. Sie ist *eine* weitere Form der Loop-Kontroll-Strukturen.

10.3.2.1 Syntax 1

Syntax für die FOR-EACH-Kontroll-Struktur:

```
FOR EACH Variable IN Expression
  <Anweisung(en)>
NEXT
```

10.3.2.2 Hinweise zur Syntax

- Wiederholt eine Folge von Anweisungen, solange die (Schleifen-)Variable nicht leer ist.
- Der Ausdruck muss ein Verweis auf ein *aufzählbares* Objekt sein – wie zum Beispiel ein Array oder eine Collection oder ein Gitter-Objekt.

10.3.2.3 Beispiel 1

Im Projekt GBTT (GamBas-ToolTipps) werden in einer Such-Funktion alle Datei-Pfade von den Dateien in einem Array gespeichert, in denen der Suchbegriff gefunden wurde. Anschließend werden zu Kontrollzwecken – in der Erprobung des Projekts – alle gespeicherten Datei-Pfade in der Konsole der Gambas-IDE angezeigt:

```
  ...
  aSuchDateiArray.Sort(0)

' Zur Kontrolle:
  FOR EACH sDateiPfad IN aSuchDateiArray
    PRINT aSuchDateiArray[iCount]
  NEXT
```

10.3.2.4 Beispiel 2

Zuerst werden alle Ausgaben des (Konsolen-)Befehls 'df' in der Variable *sResult* gespeichert. Dann werden alle Zeilen von sResult aus einem temporärem Array (→ Split(sResult,"\n")) ausgelesen. Abschließend wird jede Zeile des Arrays mit den Scan-Befehl anhand des vorgegebenen Musters in einzelne Elemente aufgeteilt und mit dem hier frei gewählten Pipe-Zeichen als Trennzeichen angezeigt:

```
DIM sResult, sLine, sElement AS String

SHELL "df" TO sResult

FOR EACH sLine IN Split(sResult, "\n")
  FOR EACH sElement IN Scan(sLine, "* * * * *")
    PRINT sElement; "|";
  NEXT ' sElement
  PRINT
NEXT ' sLine
```

Ausgabe in der Konsole:

```
Dateisystem | 1K-Blöcke | Benutzt | Verfügbar | Verw% Eingehängt auf |
/dev/sda6 | 50395844 | 9446400 | 38389444 | 20% / |
udev | 4037712 | 4 | 4037708 | 1% /dev |
tmpfs | 1618612 | 908 | 1617704 | 1% /run |
none | 5120 | 0 | 5120 | 0% /run/lock |
none | 4046524 | 156 | 4046368 | 1% /run/shm |
/dev/sda7 | 218656644 | 53868032 | 153681492 | 26% /home |
```

10.3.2.5 Beispiel 3

In Anlehnung an die Arbeitsweise der gb.settings-Komponente (→ 19.1 Settings) soll hier eine Funktion vorgestellt werden, die eine *Collection* in der Art reduzieren kann, so dass lediglich jene (Schlüssel, Wert)-Paare erhalten bleiben sollen, bei denen der Schlüssel mit einem vorgegebenen String (Präfix) beginnt.

```
Private Function Constrain(cColl As Collection, sPrefix As String) As Collection
  Dim cConstr As New Collection
```

```
  Dim vValue As Variant

  For Each vValue In cColl
  ' Elemente ueberspringen, deren Schluessel nicht mit dem gewuenschten Praefix beginnen
    If cColl.Key Not Begins sPrefix Then Continue
    cConstr[cColl.Key] = vValue
  Next
  Return cConstr

End
```

Als Anwendung können Sie folgenden Quelltext ansehen:

```
  Dim cColl As Collection = ["Kontext1/S1": "wert1", "Kontext1/S2": Pi, "Kontext2/S1": "wert2"]
  Dim vValue As Variant

  For Each vValue In Constrain(cColl, "Kontext1/")
    Print vValue
  Next
```

Ausgabe in einer Konsole:

```
wert1
3.14159265358979
```

10.3.2.6 Syntax 2

Hier finden Sie die Syntax für eine weitere Form der FOR-EACH-Kontroll-Struktur:

```
FOR EACH Expression
  <Anweisung(en)>
NEXT
```

10.3.2.7 Hinweis zur Syntax 2

- Die Syntax 2 verwenden Sie für den Fall, dass ein aufzählbares Objekt vorliegt, das aber kein realer Container ist. Der Zugriff auf den aktuellen Wert in der Aufzählung findet dann i.d.R. über Eigenschaften und Methoden des aufgezählten Objektes statt.
- Dieser o.a. Fall liegt zum Beispiel bei einem Ergebnis einer Datenbank-Abfrage (DB-Result) vor.

10.3.2.8 Beispiele für die Syntax 2

Das Beispiel bezieht sich auf eine SQLite3-Tabelle in einer SQLite3-Datenbank, in der Kontakte gepflegt werden. Es werden – zum Beispiel für einen Programmtest – von allen Kontakten nur der Vorname, der Nachname und der Wohnort in der Konsole der IDE angezeigt, obgleich alle Kontakt-Daten in der SQL-Anweisung angefordert wurden:

```
Public Sub btnShowContacts_Click()
  Dim sSQL_Anweisung As String

  DataSource1.Table = "kontakte"
  MDataBase.rDBResult = Null

  MDataBase.cDBVerbindung.Begin
    sSQL_Anweisung = "SELECT * FROM " & DataSource1.Table
    MDataBase.rDBResult = MDataBase.cDBVerbindung.Exec(sSQL_Anweisung)
  MDataBase.cDBVerbindung.Commit

  For Each MDataBase.rDBResult
   Print MDataBase.rDBResult!Vorname; " "; MDataBase.rDBResult!Nachname; " - "; MDataBase.rDBResult!Wohnort
  Next

End ' btnShowContacts_Click()
```

- Hinweise zum eingesetzten *!-Operator* finden Sie im zweiten Abschnitt von Kapitel '8.6 Spezielle Operatoren'.
- So werden in der Konsole Vorname, Nachname und Wohnort aller Kontakte formatiert angezeigt:

```
Arno Adler – Aachen
Bruno Bär - Berlin
Heinz Hirsch – Hamburg
...
```

```
Willi Wiesel – Wiesbaden
Zora Zobel - Zornheim
```

Beispiel 2

So speichern Sie beispielsweise alle benutzten Schlüssel einer Collection in einem Array:

```
Private Function CollectionKeys(cColl As Collection) As String[]
  Dim aResult As New String[]

  For Each cColl
    aResult.Add(cColl.Key)
  Next

  Return aResult

End
```

Die Eigenschaft *Collection.Key* wird bei der Enumeration der Collection auf den Schlüssel-String des aktuellen Werts gesetzt. Da der aufgezählte Wert (aus der o.a. Syntax 1) für diese Operation irrelevant ist, wird hier die zweite Syntax benutzt. Ein Aufruf der folgenden Print-Anweisung:

```
Dim cColl As Collection = ["test": "wert1", "schluessel": "wert2"]

Print CollectionKeys(cColl).Join(", ")
```

… zeigt anschließend:

```
test, schluessel
```