10.3.1 Do .. Loop

Syntax:

DO [WHILE Condition] . .. [BREAK | CONTINUE] . .. LOOP [UNTIL Condition]

Mit DO .. LOOP wird eine Reihe von Anweisungen wiederholt, so lange die Anfangsbedingung wahr bleibt oder bis die letzte Bedingung wahr wird. Wenn in der Kontrollstruktur weder WHILE noch UNTIL verwendet werden, dann ergibt sich eine Endlosschleife, die nur über die BREAK-Anweisung verlassen werden kann. Fehlt selbst die Break-Anweisung, ergibt sich eine Endlosschleife, die Sie nicht mehr stoppen können.

10.3.1.1 Syntaxbeschreibung

In der folgenden Tabelle werden die einzelnen Teil der Kontrollstruktur Do .. Loop beschrieben:

Teil	Beschreibung
Do	Die erste Anweisung in der Kontrollstruktur.
While	Wenn While verwendet wird, dann werden die Anweisungen im Schleifenkörper nur ausgeführt, wenn der nachfolgende boolsche Ausdruck TRUE ist.
Until	Wenn Until verwendet wird, dann werden die Anweisungen im Schleifenkörper nur ausgeführt, bis der nachfolgende boolsche Ausdruck TRUE ist.
Condition	Boolscher Ausdruck
Break	Sprung aus der Kontrollstruktur Do .. Loop. Die Ausführung des Programms wird mit der nächsten Zeile *nach der Kontrollstruktur Do .. Loop* fortgesetzt.
Continue	Alle Anweisungen nach Continue werden ignoriert und ein neuer Durchlauf beginnt.
Loop	Die letzte Anweisung in der Kontrollstruktur.

Tabelle 10.3.1.1.1: Beschreibung der Elemente von Do .. Loop

10.3.1.2 Beispiele

Beispiel 1

```
Public Sub LVW2C(lv As ListView, c As Collection) ' ListView To Collection
  If lv.Count > 0 Then
     c.Clear
     lv.MoveFirst()
     DO
       c[lv.Item.Key] = lv.Item.Text
     LOOP UNTIL lv.MoveNext()
  Else
    Message.Info("Die Senderliste ist leer!")
    c.Clear
    Return
  Endif
End ' LVW2C(...)
```

Beispiel 2

```
Private Sub Variante_3(iAnfang As Integer, iEnde As Integer)
  Dim iPartialsumme As Integer = 0
  Dim iSummand As Integer = iAnfang

  DO WHILE (iSummand < iEnde + 1)
     iPartialsumme = iPartialsumme + iSummand
     If IsIntegerRange(iPartialsumme) = False And b = False Then
        txtPartialsumme.Text = "FEHLER! SUMME " & String.Chr(8713) & " INTEGER"
        Return
     Endif ' Partialsumme-Überlauf ?
     Inc iSummand
  LOOP

  txtPartialsumme.Text = Str(iPartialsumme)
```

```
' Alternative Variante 3.2:
' WHILE (iSummand < iEndzahl + 1)
'   iPartialsumme = iPartialsumme + iSummand
'   INC iSummand
' WEND ' While End
' tboxPartialsumme.Text = Str(iPartialsumme)

End ' Variante_3 - Kontrollstruktur DO..WHILE..LOOP
```

10.3.1.3 Syntactic Sugar

In der Kontrollstruktur DO .. LOOP kommen viele Anweisungen vor (While, Until, Break, Continue), die Sie bereits kennengelernt haben. In Anlehnung an 'http://de.wikipedia.org/wiki/Syntactic_Sugar' versteht man unter 'syntaktischem Zucker' *Syntaxerweiterungen* in einer Programmiersprache. Diese Erweiterungen sind alternative Schreibweisen, die jedoch die Funktionalität einer Programmiersprache nicht erweitern. *Syntaktischer Zucker* lässt sich durch Änderung der Diktion auf elementare Anweisungen der Sprache zurückführen, denn

```
Do While bedingung
  ' Anweisungsfolge
Loop
```

ist eine lange Version von While-Wend; genauso, wie

```
Do
  ' Anweisungsfolge
Loop Until bedingung
```

als lange Version von Repeat-Until. Auch

```
Do
  ' Anweisungsfolge
Loop
```

produziert eine Endlosschleife, genauso wie

```
While True
  ' Anweisungsfolge
Wend.
```

Das bedeutet aber, dass man auf die Verwendung der Kontrollstruktur DO .. LOOP verzichten kann, die man in Gambas als "syntactic sugar" einstuft. Im Beispiel 2 wurde das Vorgehen mit der Alternative 3.2 demonstriert.