7.5.0 Collection

Die Klasse *Collection* (gb) implementiert eine Hash-Tabelle, deren Elemente aus einem Wert-Schlüssel-Paar bestehen.

- Die Schlüssel sind generell vom Typ *String* und die assoziierten Werte vom Daten-Typ *Variant*.
- Eine Collection ist als Daten-Sammlung oder Daten-Container aufzufassen.
- Es wird NULL als Wert verwendet, wenn kein Element einem bestimmten Schlüssel zugeordnet ist. Folglich ist das Zuordnen von NULL für einen gegebenen Schlüssel das gleiche, als würden die Daten aus der Collection entfernt.
- Die Größe der internen Hash-Tabelle wächst dynamisch mit der Anzahl der eingefügten Elemente.
- Die Klasse *Collection* verfügt über zwei Eigenschaften und fünf Methoden.
- Generell benutzt man eine Collection, wenn Daten in einem Container über Namen statt fortlaufender natürlicher Zahlen identifiziert werden sollen, wie das zum Beispiel bei Arrays der Fall ist.

7.5.0.1 Eigenschaften

Eigenschaften der Klasse *Collection*:

Eigenschaft	Datentyp	Beschreibung
Count	Integer	Gibt die Anzahl der Elemente zurück, die in einer Collection gespeichert sind.
Length	Integer	Gibt die Anzahl der Elemente zurück, die in einer Collection gespeichert sind. *Count* und *Length* werden synonym verwendet.
Key	String	Gibt den Schlüssel des zuletzt gelesenen oder aufgezählten Elements in einer Collection zurück.

Tabelle 7.5.0.1.1 : Eigenschaften der Klasse Collection

7.5.0.2 Methoden

Die Klasse *Collection* besitzt die folgenden Methoden:

Methode	Rückgabetyp	Beschreibung
Add (Value As Variant, Key As String)	–	Fügt ein Element – bestehend aus einem Wert und einem Schlüssel – in eine Collection ein.
Clear()	–	Löscht den Inhalt einer Collection.
Copy() As Collection	Collection	Die Funktion gibt eine 1:1-Kopie der originalen Collection zurück.
Exist (Key As String) As Boolean	Boolean	Die Funktion gibt True zurück, wenn es zu dem vorgegebenen Schlüssel einen an diesen Schlüssel gebundenen Wert gibt.
Remove (Key As String)	–	Entfernt das Element mit dem vorgegebenen Schlüssel *Key* aus einer Collection – falls das Element existiert, was mit der Methode *Collection.Exist(Key)* sicher geprüft werden kann.

Tabelle 7.5.0.2.1 : Methoden der Klasse Collection

7.5.0.3 Erzeugen einer Collection

Es gibt unterschiedliche Möglichkeiten um eine Collection zu erzeugen:

(A) Deklaration einer Variablen vom Daten-Typ *Collection* und *spätere* Zuweisung der Elemente
(B) Deklaration einer Variablen vom Daten-Typ *Collection* mit *direkter* Wertzuweisung
(C) Inline-Collection erzeugen
(D) Kopie einer Collection

7.5.0.4 Beispiele

Die folgenden Beispiele setzen die im voran gegangenen Kapitel aufgeführten Möglichkeiten um.

(A) Deklaration einer Variablen vom Daten-Typ *Collection* und *spätere* Zuweisung der Elemente

```
Dim hCollection As Collection

hCollection = New Collection ( [ Mode As Integer ] )
```

Es wird eine neue Collection erzeugt. Die Angabe von Mode ist optional. Der Modus *gb.binary* ist der Standard. Der Modus beschreibt die verwendete Methode für den Vergleich von Schlüsseln und es gilt unter Verwendung der zwei Gambas-Konstanten:

```
gb.Binary:
Diese Konstante repräsentiert 0 und gibt an, dass Groß- und Kleinschreibung beachtet wird.
gb.IgnoreCase:
Diese Konstante repräsentiert 1 → Groß- und Kleinschreibung wird nicht beachtet.
```

Beispiel 1

```
Public Sub btnAddMethod_Click()
  Dim cCollection As New Collection(gb.IgnoreCase)
  Dim vValue, vElement As Variant
  Dim aNames As String[]
  Dim icount As Integer

  aNames = ["Adam", "Ben", "Charlie"]

' cCollection.Add(Value As Variant, Key As String) → Zuerst der Wert und dann der Schlüssel
  cCollection.Add("a", "a")
  cCollection.Add(3, "A")
  cCollection.Add(aNames, "v") ' Der Wert ist ein String-Array
  cCollection.Add(False, "d")
  cCollection.Add(8.44, "e")
  cCollection.Add(Format(Date(Now), "d. mmmm yyyy"), "f")

  For Each vValue In cCollection
    If cCollection.Key = "v" Then
       For iCount = 0 To vValue.Max
         Print cCollection.Key & (icount + 1); " --> "; vValue[iCount]
       Next ' iCount
    Else
       Print cCollection.Key; " --> "; vValue
    Endif ' cCollection.Key = "v" ?
  Next ' vValue

End ' btnAddMethod_Click()
```

Die Ausgabe in der Konsole der Gambas-IDE demonstriert auch das Auslesen der mit dem Schlüssel assoziierten Werte:

```
a  --> 3
v1 --> Adam
v2 --> Ben
v3 --> Charlie
d  --> False
e  --> 8,44
f  --> 7. April 2014
```

Die Groß- und Kleinschreibung des Schlüssels wird in diesem Modus *ignoriert*. Das erste Element wird deshalb überschrieben. Aus dem Key "A" wird "a" mit dem Wert 3, weil für den gleichen Schlüssel *zwei* Werte notiert sind.

Damit findet auch die Frage "Kann man in einer Collection unterschiedliche Daten-Typen für die Werte verwenden?" eine klare Antwort: "Ja - jeder Wert einer Collection ist vom Typ Variant und ein Variant kann alles tragen – unabhängig von der Collection und unabhängig von den anderen Werten in einer Collection".

Beispiel 2

Im Beispiel 1 könnte man die Zuweisung der Werte zu den Schlüsseln auch alternativ so vornehmen:

```
' cCollection[Key As String] = Value As Variant → Zuerst der Schlüssel und dann der Wert
  cCollection["a"] = "a"
  cCollection["A"] = "3"
  cCollection["v"] = aNames ' Der Wert ist ein String-Array
  cCollection["d"] = False
  cCollection["e"] = 8.44
  cCollection["f"] = Format(Date(Now), "d. mmmm yyyy")
```

Hinweis:

Der !-Operator ist ein spezieller Operator für Container-Objekte, die Zugriff auf ihre Elemente anhand eines Schlüssels vom Typ 'String' erlauben → Kapitel 8.6. Eine Collection gehört auch dazu. Dabei wird der Schlüssel-String nach dem !-Operator notiert, so dass zum Beispiel *cCollection!Name* gleichbedeutend ist mit *cCollection["Name"]*. Beachten Sie, dass die Anführungszeichen um den Schlüssel-String beim Einsatz des !-Operators entfallen. Der Rückgabewert ist in beiden Fällen der Wert in der Collection mit dem Schlüssel "Name".

```
sCurrent.Name = cCollection["Name"]
sCurrent.Name = cCollection!Name ' Alternative Schreibweise
```

(B) Inline-Collection

Seit der Revision #1699 (November 2008) gibt es in Gambas diese neue Syntax zum Anlegen einer Collection. Es wird bei dieser kompakten *Inline-Syntax* erst der Schlüssel notiert und dann der Wert:

```
  Dim cNames, cNamesCopy As New Collection
  Dim vValue, vElement As Variant

' Syntax: Collection = [ Key: Expression [ , ... ] ]
  cNames = ["w": ["Anna", "Brit", "Claudia", "Doreen"],
            "m": ["Adam", "Bruno", "Clemens"],
            "Nachname": ["Adler", "Fuchs", "Katze", "Wiesel", "Zebra"]]

' Ausgabe in der Konsole der Gambas-IDE
  For Each vValue In cNames
      For Each vElement In cNames[cNames.Key]
          Print cNames.Key; " -> "; vElement
      Next ' vElement
  Next ' vValue
```

(C) Deklaration einer Variablen vom Daten-Typ *Collection* mit *direkter* Wertzuweisung.

Diese Variante nutzt die *Inline-Syntax*:

```
Dim cCollection As Collection = ["Blue": &H0000FF&, "White": &HFFFFFF&, "Red": &HFF0000&]
```

(D) Kopie einer Collection

Wenn Sie eine Kopie einer existierenden Collection mit der Methode *Collection.Copy()* erzeugen, ist diese Kopie ein *eigenständiges* Collection-Objekt:

```
  Dim cNames, cNamesCopy As New Collection

  cNames = ["w": ["Anna", "Brit"], "m": ["Adam", "Bruno"], "Nachname": ["Adler", "Katze"]]
  cNames["Nachname"].Add("Maus", 1) ' Einfügen nach dem Adler (Index = 0)...

  cNames.Name = "cNAME"

  If cNames.Count > 0 Then
     cNamesCopy = cNames.Copy()
  Else ' Count = 0
     Message.Error(„Die Collection '" & cNames.Name & „' hat keine Elemente")
  Endif ' cNames.Count > 0
```

Die Anweisung *cNames.Name = "cNAME"* wird im Abschnitt 7.5.0.7 näher beschrieben.

7.5.0.5 Zugriff auf Elemente in einer Collection

Sie können auf *einzelne* Werte einer Collection zugreifen, was die genaue Kenntnis des Schlüssels voraussetzt oder auf alle Werte:

```
  Dim cColor As New Collection
  Dim vValue As Variant

  cColor.Add("&HC3DDFF", "TFarbe")
  cColor.Add("&HD6D4D2", "HFarbe")
  cColor.Add("&HF5FFE6", "EFarbe")

  Print "Die Hintergrund-Farbe ist "; cColor["HFarbe"] ' Ein Element anzeigen
' Print "Die Hintergrund-Farbe ist "; cColor!HFarbe → Alternative

  For Each vValue In cColor
    Print cColor.Key, String.Chr(187), vValue ' Alle Elemente (Schlüssel-Wert-Paare) anzeigen
  Next ' vValue
```

7.5.0.6 Export und Import einer Collection

Es wird ein *gambas-spezifisches* Speicher-Management verwendet, mit dem Sie eine Collection in einer Datei abspeichern können (Daten-Export) oder den Inhalt einer Datei in eine Collection einlesen können (Daten-Import). Weitere Informationen finden Sie im Kapitel 7.2.2.3 Daten-Export und Daten-Import.

Daten-Export
Die folgende Prozedur speichert eine Collection in einer binären, gambas-spezifischen Datei:

- Parameter1: Datei-Pfad zur Export-Datei
- Parameter2: Referenz auf die ausgewählte Collection

```
Public hFile As File
Public sFilePath As String = Application.Path &/ "url.list"
Public cData As New Collection

Public Sub Form_Open()
...
  If Exist(sFilePath) Then
     cData = ImportCData(sFilePath)
     For Each vValue In cData
         lsbURL.Add(cData.Key)
     Next
  Else
     cData["http://mp3channels.webradio.rockantenne.de/classic-perlen"] = "ROCK ANTENNE"
     lsbURL.Add("http://mp3channels.webradio.rockantenne.de/classic-perlen")
  Endif ' Exist(sFilePath) ?
End ' Form_Open()

Public Sub ExportCData(sPath As String, cExport As Collection)

  If cExport.Count = 0 Then
     Return
  Else
     hFile = Open sPath For Write Create ' Die Exportdatei wird stets neu angelegt
     Write #hFile, cExport As Collection
     Close #hFile
     Catch
     Message.Error("Der Daten-Export war fehlerhaft!" & gb.NewLine & "Fehler: " & Error.Text)
  Endif ' c.Count = 0 ?

End ' ExportCData(...)
```

Daten-Import
Diese u.a. Funktion importiert den Inhalt einer Collection aus einer binären, gambas-spezifischen Datei und gibt als Funktionswert eine Collection zurück:

- Parameter: Datei-Pfad zur Import-Datei

```
Public Function ImportCData(sPath As String) As Collection
  Dim cTempImport As Collection

  hFile = Open sPath For Read
  cTempImport = Read #hFile As Collection
  Close #hFile
  Return cTempImport
  Catch
  Message.Error("Der Daten-Import war fehlerhaft!" & gb.NewLine & "Fehler: " & Error.Text)

End ' ImportCData(...)
```

7.5.0.7 Erweiterung der Klasse 'Collection'

Möchten Sie, dass eine Collection einen Namen bekommt – den sie p.d. nicht besitzt – dann erweitern Sie die Collection-Klasse, indem Sie eine Klasse 'Collection.class' in ihr Projekt einfügen, diese exportieren und dann die *neue* Eigenschaft 'Name' zur Klasse Collection hinzufügen – die originale Klasse wird erweitert. Hier folgt der Inhalt der neu angelegten Klassen-Datei 'Collection.class':

```
' Gambas class file

Export
Property Name As String
Private $sName As String

Private Sub Name_Read() As String
  Return $sName
End

Private Sub Name_Write(Value As String)
  $sName = Value
End
```

Jetzt können Sie jeder Collection in ihrem Projekt einen Objekt-Namen zuweisen und diesen im Programm abfragen. Nur so sind informative Meldungen der folgenden Art möglich:

```
Dim cURL As New Collection

cURL.Name = "TX_URL"

If cURL.Count = 0 Then
   Message.Warning("Die Collection '" & cURL.Name & "' existiert, ist aber leer!")
Endif ' cURL.Count = 0 ?
```

Details zur Erweiterung oder Änderung von Klassen finden Sie im Kapitel 26 → Klassen, Module und Bibliotheken.