24.6.9.1 Projekt Formatierer für JSON-Text

Es werden Ihnen zukünftig mehr Anwendungen begegnen, die als Format für den Export oder Import von Daten auf das kompakte JSON-Format setzen. Die Daten liegen serialisiert als JSON-Text vor.

Für die *Verarbeitung* von Daten im JSON-Format realisiert die Methode JSON.Decode(...) sowohl einen Validator als auch einen Parser für einen JSON-Text. Aus den Daten im JSON-Format werden Daten vom Gambas-Typ Variant und können vom Typ Collection oder JSONCollection sein.

Der folgende JSON-Text mit Wetter-Daten zum Beispiel ist sehr kompakt:

```
{"coord":{"lon":11.77,"lat":52.78},"weather":[{"id":801,"main":"Clouds","description":"few  clouds","icon":
"02d"}],"base":"cmc stations","main":{"temp":12.48,"pressure":1017.3,"humidity":85,"temp_min":12.48,
"temp_max":12.48,"sea_level":1021.8,"grnd_level":1017.3},"wind":{"speed":6.67,"deg":242.504},"clouds":
{"all":24},"dt":1459947946,"sys":{"message":0.0075,"country":"DE","sunrise":1459917197,"sunset":
1459965476},"id":2856639,"name":"Osterburg","cod":200}
```

Wesentlich besser lesbar ist der *gleiche* Text, wenn er formatiert ausgegeben wird. Da der formatierte JSON-Text gut strukturiert wird, ist er auch gut lesbar:

```
{
  coord: {
    lon: 11,77,
    lat: 52,78
  },
  weather: [
    {
      id: 801,
      main: "Clouds",
      description: "few clouds",
      icon: "02d"
    }
  ],
  base: "cmc stations",
  main: {
    temp: 12,48,
    pressure: 1017,3,
    humidity: 85,
    temp_min: 12,48,
    temp_max: 12,48,
    sea_level: 1021,8,
    grnd_level: 1017,3
  },
  wind: {
    speed: 6,67,
    deg: 242,504
  },
  clouds: {
    all: 24
  },
  dt: 1459947946,
  sys: {
    message: 0,0075,
    country: "DE",
    sunrise: 1459917197,
    sunset: 1459965476
  },
  id: 2856639,
  name: "Osterburg",
  cod: 200
}
```

Für die formatierte *Anzeige* von Daten im JSON-Format existiert keine Methode in Gambas. In diesem Projekt werden Ihnen daher zwei Varianten vorgestellt, wie Sie einen JSON-Text formatiert ausgeben können.

- Variante 1:
  Die JSON-Grammatik im RFC 7159 (→ https://tools.ietf.org/html/rfc7159) ist sehr einfach. Es bietet sich an, bei der Analyse der Struktur eines JSON-Textes diese Grammatik als Vorlage zu nehmen, weil sie die Struktur von JSON-Text eindeutig beschreibt.

- Variante 2:
  Einsatz des JSON-Tools der Sprache Python über einen Shell-Aufruf.

24.6.9.1.1 Variante 1

Im RFC-Text wird gesagt, dass ein JSON-Wert (value) so aufgebaut ist:

```
value = object | array | number | string | 'true' | 'false' | 'null'
```

Ein Array zum Beispiel wird nach der Grammatik so beschrieben:

```
array = "[" [value *("," value)] "]"
```

Diese Beschreibung lässt sich neben den anderen in die folgenden Gambas-Prozeduren übertragen:

```
Private Sub PutValue()
  txaOutput.Insert(Space$(iIndent) & IIf(sKey, "\"" & sKey & "\"" & ": ", ""))
  Select Case TypeOf(vValue)
    Case gb.Object
      If vValue Is JSONCollection Then
         PutObject()
      Else If vValue Is Variant[] Then
         PutArray()
      Endif
    Case gb.Integer, gb.Long, gb.Float
      PutNumber()
    Case gb.String
      PutString()
    Case gb.Boolean
      PutBoolean()
    Case gb.Null
      PutNull()
  End Select
End ' PutValue()

Private Sub PutObject()
  Dim vElement As Variant, iElement As Integer, jCol As JSONCollection = vValue

  txaOutput.Insert("{" & gb.NewLine)
  iIndent += 2
  iElement = 1
  For Each vElement In jCol
    sKey = jCol.Key
    vValue = vElement
    PutValue()
    txaOutput.Insert(IIf(iElement < jCol.Count, ",", "") & gb.NewLine)
    Inc iElement
  Next
  iIndent -= 2
  txaOutput.Insert(Space$(iIndent) & "}")

End ' PutObject()

Private Sub PutArray()
  Dim iIndex As Integer, aArray As Variant[] = vValue

  txaOutput.Insert("[" & gb.NewLine)
  iIndent += 2
  For iIndex = 0 To aArray.Max
    sKey = ""
    vValue = aArray[iIndex]
    PutValue()
    txaOutput.Insert(IIf(iIndex < aArray.Max, ",", "") & gb.NewLine)
  Next
  iIndent -= 2
  txaOutput.Insert(Space$(iIndent) & "]")

End ' PutArray()

Private Sub PutBoolean()
  txaOutput.Insert(IIf(vValue, "true", "false"))
End ' PutBoolean()

Private Sub PutNumber()
  txaOutput.Insert(Str$(vValue))
End ' PutNumber()

Private Sub PutString()
  txaOutput.Insert(Subst$("\"&1\"", vValue))
End ' PutString()

Private Sub PutNull()
  txaOutput.Insert("null")
End ' PutNull()
```

Beachten Sie, dass der Projekt-Quelltext die JSON-Grammatik 1:1 abbildet. Ein Nicht-Terminal-Symbol in der Grammatik – zum Beispiel String – korrespondiert immer mit einer Funktion, die das Nicht-Terminal-Symbol erkennt und ausgibt, wobei die Funktionen sich gegenseitig aufrufen.

Der Prozedur EditData(...) wird als Parameter der JSON-Text übergeben, der dekodiert wird und dann in der globalen Variablen vValue der Prozedur PutValue() zur Verfügung steht:

```
Private Sub EditData(Data As String)

  Dim jCollection As JSONCollection

  Try jCollection = JSON.Decode(Data, True) ' VALIDATOR

  If Error Then
     Message.Error("<b>Error decoding!</b><br>" & Error.Text & "  Error-Code = " & Error.Code)
     Return
  Endif

  sKey = Null
  vValue = jCollection
  iIndent = 0
  PutValue()

End ' EditData(Data As String)
```

Ausgelöst wird die strukturierte Anzeige des JSON-Textes durch die folgende Prozedur:

```
Public Sub btnFormattingJSON_Click()
  txaOutput.Clear()
  EditData(txaInput.Text)
End ' btnFormattingJSON_Click()
```

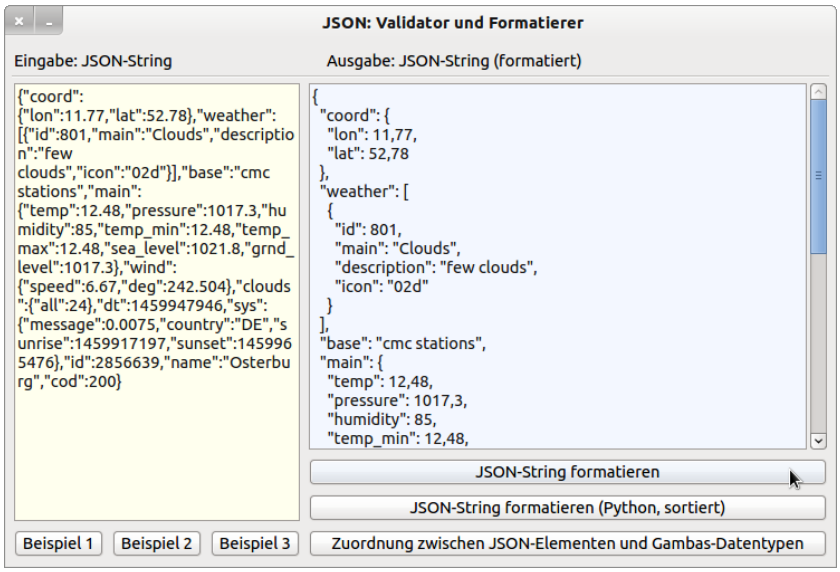

Abbildung 24.6.9.1.1: Formatierte Anzeige – Variante 1

Die Anzeige des formatierten JSON-Textes erfolgt, wenn der JSON-Text syntaktisch in Ordnung ist.

```
Try jCollection = JSON.Decode(Data, True) ' VALIDATOR
```

Tritt beim Dekodieren des JSON-Textes ein Fehler auf, so wird dieser angezeigt – das Projekt enthält somit auch einen Validator für JSON-Text:

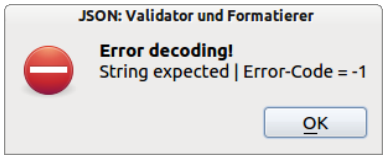

Abbildung 24.6.9.1.2: Fehler-Meldung (Validator)

Das Projekt-Beispiel 3 zeigt den JSON-Text für die Daten einer TreeView. Im Projekt zum → Kapitel 17.9 TreeView werden die Daten im JSON-Format exportiert und importiert. Das ermöglicht es während der Erprobung des Projektes jederzeit auf die exportierten Elemente der TreeView zuzugreifen und aus den importieren Daten die Elemente der TreeView zu generieren und anzuzeigen:

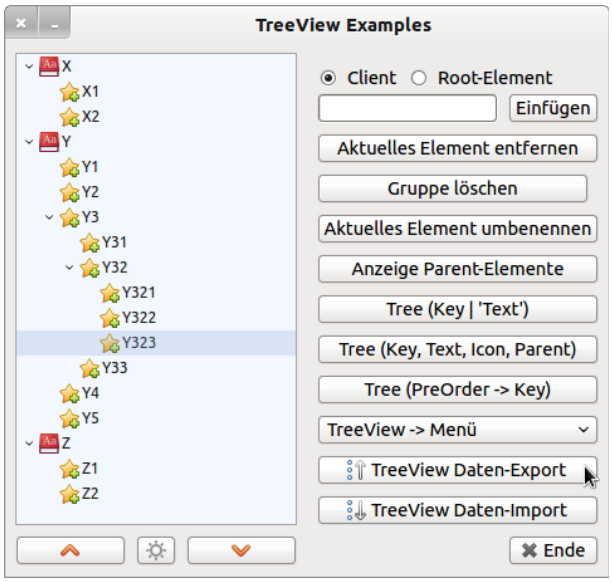

Abbildung 24.6.9.1.3: GUI zur Erprobung von Eigenschaften und Methoden einer TreeView

24.6.9.1.2 Variante 2

Die formatierte Ausgabe von JSON-Text – nach den einzelnen Schlüsselwerten sortiert – können Sie auch so erreichen:

```
Public Sub btnFormattingJSONPython_Click()

 txaOutput.Clear()
 Try Shell "echo '" & txaInput.Text & "' | python -m json.tool" To txaOutput.Text
 txaOutput.Pos = 0

End ' btnFormattingJSONPython_Click()
```

24.6.9.1.3 Erweiterung

In das Projekt floss eine Erweiterung ein, mit deren Hilfe Sie die Zuordnung zwischen Schlüssel und Wert-Typ des dekodierten JSON-Textes (Typ Collection oder JSONCollection) ermitteln und anzeigen können. Den Vorteil des Einsatzes der Prozedur GetStructure(...) und der Funktion GetType(...) erkennen Sie erst dann, wenn Sie das Projekt im → Kapitel 24.6.9.2 erproben:

```
Private Sub GetStructure(Data As String)

  Dim i As Integer = 1, vVariant As Variant
  Dim jCollection As JSONCollection

  Try jCollection = JSON.Decode(Data, True)
  If Error Then
     Message.Error("<b>Error decoding!</b><br>" & Error.Text & "  Error-Code = " & Error.Code)
     Return
  Endif

  txaOutput.Insert(gb.NewLine)
  txaOutput.Insert("Anzahl der Elemente in der JSON-Collection: " & jCollection.Count & gb.NewLine)
  txaOutput.Insert(gb.NewLine)
  txaOutput.Insert(String$(100, "-") & gb.NewLine & gb.NewLine)

  For Each vVariant In jCollection
    txaOutput.Insert(("Key ") & Str(i) & " : \"" & jCollection.Key & "\"" & (" ---> Value-Type: ") & /
                     GetType(vVariant) & gb.NewLine)
    Inc i
  Next
End ' GetStructure(...)
```

```
Private Function GetType(Value As Variant) As String
  Select Case TypeOf(Value)
    Case gb.Boolean ' 1
      Return "Boolean"
    Case gb.Integer ' 4
      Return "Integer"
    Case gb.Long ' 5
      Return "Long-Integer"
    Case gb.Float ' 7
      Return "Float"
    Case gb.String '9
      Return "String"
    Case gb.Null ' 15
      Return "NULL OR JSON-Null"
    Case gb.Object ' 16
      If Value Is JSONCollection Then Return "JSONCollection"
      If Value Is Collection Then Return "Collection"
      If Value Is Variant[] Then Return "Variant[]"
  End Select
  Return "?"
End ' GetType(...)
```

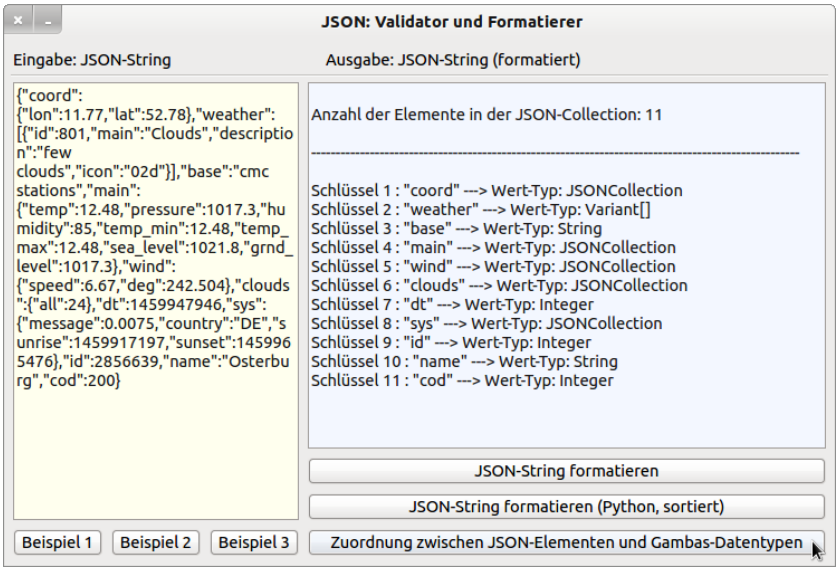

Abbildung 24.6.9.1.4: Zuordnung zwischen Schlüssel und Wert-Typ