24.9.0.3 D-Bus-Signatur

In Gambas kann eine Methode – als Funktion deklariert – genau einen Wert zurückgeben. Ein D-Bus-Signal oder eine D-Bus-Methode dagegen können mehrere Werte zurückgeben. Über die Signatur eines Signals oder einer Methode erkennen Sie auf einfache Art, wie viel Werte mit einem empfangenen Signal oder mit einem Methoden-Aufruf über deren Argumente – gespeichert in einem Variant-Array – zurückgegeben werden und welchen Daten-Typ die einzelnen D-Bus-Werte haben.

Die Klassen *DBusVariant* und *DBusValues* (gb.dbus) ermöglichen es Ihnen, D-Bus-Werte zu einer vorgegebenen D-Bus-Signatur zu definieren. Die Beschreibung der beiden Klassen *DBusVariant* und *DBusValues* können Sie im Kapitel 24.9.8.0 nachlesen. Die beiden Klassen besitzen nur die Eigenschaft Value und jeweils eine Konstante. Dort erfahren Sie auch, dass Sie bei einer Signatur wie zum Beispiel "(ia{sv}av)" die Klasse DBusVariant und bei einer Signatur wie "u(ia{sv}av)" die Klasse DBusValues einsetzen müssen.

Hinweis: Unter http://gambaswiki.org/wiki/doc/dbus#t10 finden Sie eine Tabelle mit einer Zusammenfassung, wie Gambas-Datentypen in D-Bus-Datentypen konvertiert werden und umgekehrt.

Festlegung: Eine D-Bus-Signatur wird als String-Konstante definiert.

24.9.0.3.1 Beispiel 1

Um den Rückgabewert einer D-Bus-Methode oder eines abgefangenen D-Bus-Signals auszuwerten benötigen Sie deren Signaturen, um die D-Bus-Daten mit ihren D-Bus-Datentypen auf Gambas-Daten und deren Datentypen abzubilden.

Im Beispiel 1 wird ein spezielles Signal "MountAdded" analysiert, das von der d-bus-fähigen Anwendung mit dem D-Bus-Namen *org.gtk.vfs.UDisks2VolumeMonitor* und mit dem Objekt-Pfad */org/gtk/Private/RemoteVolumeMonitor* – neben vielen anderen Signalen – gesendet wird, wenn etwa ein USB-Stick eingesteckt wird. Sie können das Konsolen-Programm *dbus-send* für die Introspection einsetzen:

```
$ dbus-send --session --print-reply --dest=org.gtk.vfs.UDisks2VolumeMonitor \
 /org/gtk/Private/RemoteVolumeMonitor \
 org.freedesktop.DBus.Introspectable.Introspect > volumemonitor.introspection.xml
```

Hier ein Ausschnitt aus der XML-Datei volumemonitor.introspection.xml:

```
<signal name="MountAdded">
  <arg type="s" name="dbus_name"/>
  <arg type="s" name="id"/>
  <arg type="(sssssbsassa{sv})" name="mount"/>
</signal>
```

Sie benötigen die angegebenen Signaturen, um die D-Bus-Daten mit ihren D-Bus-Datentypen auf Gambas-Daten und deren Datentypen abzubilden. Nach der o.a. Konvertierungs-Tabelle erkennen Sie, dass die ersten beiden Argumente den nativen Daten-Typ "s" besitzen, den auch Gambas kennt. Daher müssen Sie hier nichts konvertieren. Völlig anders sieht es beim dritten Argument mit der Signatur "ssssssbsassa{sv}", die als komplexer Daten-Typ charakterisiert werden kann:

```
String String String String String String Boolean String String-Array String Collection(String:Variant)
```

Das sind die einzelnen Gambas-Daten-Typen, die sich nach der o.a. Tabelle für das dritte Argument angeben lassen, wobei das dritte Argument vom Typ her ein Variant-Array ist – erkennbar an den einschließenden runden Klammern in der Signatur. Die runden Klammern werden in der D-Bus-Notation auch als *Struct of (…)* notiert. Array of [datatype] kennzeichnet ein Array mit Elementen vom angegebenen Datentyp. Die D-Bus-Notation *Dict of {…}* steht zum Beispiel für a{ss} oder a{sv} und erfordert die Konvertierung in den Gambas-Daten-Typ Collection.

```
Struct of (
s                           6x String
b                           1x Boolean
s                           1x String
Array of [string]           1x String-Array
s                           1x String
Dict of {String,Variant}    1x Collection (Key-Typ=String, Value-Typ=Variant)
)
```

Wenn Sie das Signal über die Klasse DBusObserver mit dem Message-Typ 'DBus.Signal' und dem Ereignis hDbusObserver_Message() abfangen, dann bieten sich diese Deklaration passender Variablen und die nachfolgenden Wert-Zuweisungen (Variable => Argument-Wert) an:

```
Public Sub hDBusObserver_Message()

  Dim bTF as Boolean
  Dim k as Integer = 1
  Dim sA1, sA2 As String
  Dim sA31, sA32, sA33, sA34, sA35, sA36, sA37, sA38 As String
  Dim vElement As Variant
  Dim cCollection As Collection
  Dim aArray As String[]
  Dim aSignalArguments As Variant[]

  aSignalArguments = hDBusObserver.Message.Arguments ' Speichern aller Signal-Argumente in aSignalArguments

  sA1  =  aSignalArguments[0]
  sA2  =  aSignalArguments[1]
  sA31 =  aSignalArguments[2][0]
  sA32 =  aSignalArguments[2][1]
  sA33 =  aSignalArguments[2][2]
  sA34 =  aSignalArguments[2][3]
  sA35 =  aSignalArguments[2][4]
  sA36 =  aSignalArguments[2][5]
  bTF  =  aSignalArguments[2][6]
  sA37 =  aSignalArguments[2][7]

  aArray = New String[]
  aArray = aSignalArguments[2][8]
  If aArray.Count > 0 Then
     For Each vElement In Array
       Print "Element " & Str(k) & "  =    " & vElement
       Inc k
     Next
  Else
     Print "Attention: The string array is empty!" & gb.NewLine
  Endif

  sA38 =  aSignalArguments[2][9]

' Collection -> KeyType = String, DataType = Variant
  cCollection = New Collection
  cCollection = aSignalArguments[2][10]
  If cCollection.Count > 0 Then
     For Each vElement In cCollection
       Print "Element " & Str(k) & "  :    " & cCollection.Key & " = " & vElement & gb.NewLine
       Inc k
     Next
  Else
     Print "Attention: The collection is empty!"
  Endif

End
```

24.9.0.3.2 Beispiel 2

Für dieses Beispiel werden Ihnen ein D-Bus-Daten-Server und ein D-Bus-Client vorgestellt. Der Server stellt einen Dienst zur Verfügung, beim dem mit einem Methoden-Aufruf (GetT(…)) mit einem (String-) Argument ein Wert mit einem komplexen Daten-Typ zurückgegeben wird.

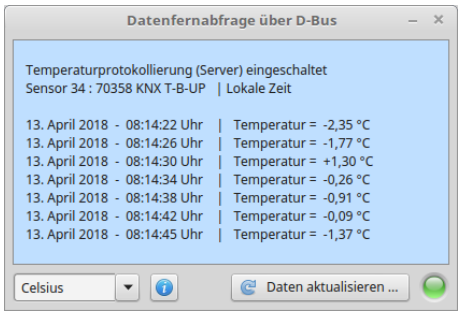

Abbildung 24.9.0.3.1: Der Service wird genutzt ...

Das zweite Beispiel ist zur Demonstration bewußt so konstruiert, dass die Signatur des Rückgabewertes viele Datentypen nutzt.

```
String          s0     ' Sensor Number/Type: 70358 KNX T-B-UP
Integer 1       i1     ' Day     15
Integer 2       i2     ' Month   12
Integer 3       i3     ' Year    2017

Variant-Array   av
Array[0]        i4     ' Hour    15
Array[1]        i5     ' Minute  24
Array[2]        i6     ' Second  37
Array[3]        s1     ' Time zone "LocalTime"

Collection      a{sv}  ' Key.Type=String, Value-Type=Variant
c["T"]          f1     ' Temperature -1.4
c["L"]          s2     ' Label       °
c["S"]          s3     ' Scale       C

Boolean         b0     ' TempLog     True/False
```

Mit diesen Vorgaben für den Rückgabewert der Methode GetT(…) ergibt sich nach einer Konvertierung der Gambas-Daten-Typen in D-Bus-Datentypen die folgende Signatur, die in einer eigenen Datei – hier RValue.class – als String-Konstante beschrieben wird:

```
' Gambas class file

Export
Inherits DBusValues

Public Const Signature As String = "siiiava{sv}b"
```

Eine Introspection für das exportierte Objekt liefert diese Übersicht:

```
  <interface name="org.gambas.dbusserver3.msservice">
    <method name="GetT">
      <arg name="arg1" type="s"/>
      <arg name="value1" type="s" direction="out"/>
      <arg name="value2" type="i" direction="out"/>
      <arg name="value3" type="i" direction="out"/>
      <arg name="value4" type="i" direction="out"/>
      <arg name="value5" type="av" direction="out"/>
      <arg name="value6" type="a{sv}" direction="out"/>
      <arg name="value7" type="b" direction="out"/>
    </method>
  </interface>
```

In einem weiteren Schritt müssen Sie nach der Deklaration der Signatur die Methode GetT(…) in einer weiteren Klassen-Datei (MSService.class) festlegen. Der Datentyp des Funktionswertes ist vom Typ RValue:

```
' Gambas class file

Inherits DBusObject
Create Static

' Signal definition ...

'' Definition of a method. The method has exactly one argument.
Public Function GetT(ShortScale As String) As RValue

  Dim hItem As RValue
  Dim s0, s1, s2, s3 As String
  Dim i1, i2, i3, i4, i5, i6 As Integer
  Dim f1 As Float
  Dim aTime As Variant[]
  Dim cTemperature As Collection
  Dim bTempLog As Boolean

  s0 = "Sensor 34 : 70358 KNX T-B-UP"

  i1 = Day(Now())
  i2 = Month(Now())
  i3 = Year(Now())

  i4 = Hour(Now())
  i5 = Minute(Now())
  i6 = Second(Now())
  s1 = ("LocalTime")
```

```
  aTime = [i4, i5, i6, s1]

' The temperature value is read out in real operation from an RS232-AD converter
' A random value for the current temperature is generated here
  f1 = Round(Rnd(-3, 2), -4)

  Select Case ShortScale
    Case "C"
      s2 = "°"
      s3 = "C"
    Case "K"
      f1 = f1 + 273.15
      s2 = ""
      s3 = "K"
    Case "F"
      f1 = (9 / 5) * f1 + 32
      s2 = "°"
      s3 = "F"
    Default
      s2 = "°"
      s3 = "C"
  End Select

  cTemperature = New Collection
  cTemperature.Add(f1, "Temperature") ' Temperature
  cTemperature.Add(s2, "Label")       ' Label
  cTemperature.Add(s3, "Scale")       ' Temperature scale

  bTempLog = True

  hItem = New RValue
' hItem.Value with complex data type:
  hItem.Value = [s0, i1, i2, i3, aTime, cTemperature, bTempLog]

  Return hItem

End
```

Im Download-Bereich finden Sie die Projekte für den D-Bus-Server3 und den D-Bus-Client3, damit Sie das Zusammenspiel von D-Bus-Signatur und Daten-Konvertierung selbst nachvollziehen können.

Der Client liest nicht nur den komplexen Rückgabewert mit der Signatur "siiiava{sv}b" aus, sondern wertet auch ein Signal aus, dass der Server sendet, wenn er ein- oder ausgeschaltet wird. Für den Client wurde zusätzlich auch eine Introspection des Server-Dienstes implementiert.