24.9.5.1 Projekt 1 – DBusObserver – Teil 1

Im vorgestellten Programm soll festgestellt werden, ob ein USB-Stick eingesteckt wurde, welchen Namen (Label) der USB-Stick hat und unter welchem Pfad der USB-Stick erreichbar ist.

Wenn Sie ein Projekt für einen D-Bus-Observer entwickeln wollen, der eine Methode oder ein Signal beobachten soll, dann müssen Sie im Vorfeld die Antworten auf die folgenden Fragen einholen:

- Welche d-bus-fähige Anwendung sendet die Nachricht, die abgefangen werden soll?
- Wird die Methode oder das Signal auf dem Session-Bus oder auf dem System-Bus gesendet?
- Welchen Typ repräsentiert die Nachricht?
- Welches D-Bus-Objekt wird verwendet?
- Ist der Objekt-Pfad bekannt?
- Welchen Namen hat die Methode oder das zu beobachtende Signal (Member)?
- Welche Signatur hat die Nachricht?
- Welchen Namen hat die Schnittstelle (Interface) – wenn eine solche existiert?

Die Antworten auf die Fragen können gegeben werden, wenn Sie die beiden Programme d-feet und dbus-monitor einsetzen und deren Ausgaben beim Einstecken eines USB-Sticks sorgfältig in Bezug auf die o.a. Aufgabenstellung analysieren. Der Einsatz des Programms d-feet wird im Kapitel D-Bus-Introspection vorgestellt.

Sehr gute Dienste leistet auch das Konsolen-Programm *dbus-send* für die Introspection:

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

Hier ein Ausschnitt aus der XML-Datei:

```
...
  <signal name="VolumeAdded">
    <arg type="s" name="dbus_name"/>
    <arg type="s" name="id"/>
    <arg type="(sssssbbssa{ss}sa{sv})" name="volume"/>
  </signal>
...
```

Ergänzend folgt ein Ausschnitt der Ausgaben des Programms *dbus-monitor*, wenn ein USB-Stick eingesteckt wurde und u.a. das dann ausgelöste D-Bus-Signal "VolumeAdded" des Objekts /org/gtk/Private/RemoteVolumeMonitor mit den Werten seiner Argumente angezeigt wird:

```
signal sender=:1.10 -> dest=(null destination) serial=53 path=/org/gtk/Private/RemoteVolumeMonitor;
   interface=org.gtk.Private.RemoteVolumeMonitor; member=VolumeAdded
   string "org.gtk.Private.UDisks2VolumeMonitor"
   string "0x2493c70"
   struct {
      string "0x2493c70"
      string "2_GB_P"
      string ". GThemedIcon drive-removable-media-usb drive-removable-media drive-removable drive"
      string ". GThemedIcon drive-removable-media-usb-symbolic ... drive-removable drive"
      string ""
      string ""
      boolean true
      boolean true
      string "0x24b4d90"
      string ""
      array [
         dict entry(
            string "class"
            string "device"
         )
         dict entry(
            string "unix-device"
            string "/dev/sdd1"
         )
         dict entry(
            string "label"
            string "2_GB_P"
         )
         dict entry(
            string "uuid"
```

```
            string "38ea6e0b-a161-401d-a673-11c06cf1229e"
         )
      ]
      string "gvfs.time_detected_usec.1519387042284411"
      array [
      ]
   }
```

Damit und auch nach einem Blick in die Ausgaben des Konsolen-Programms d-feet ergaben sich die folgenden konkreten Antworten für die zutreffende Anwendung:

- Anwendung (D-Bus-Name): org.gtk.Private.UDisks2VolumeMonitor
- D-Bus: Session-Bus
- Typ der Nachricht: Signal
- D-Bus-Objekt-Pfad: /org/gtk/Private/RemoteVolumeMonitor
- Name des Signals (Member): VolumeAdded
- Die Nachricht hat drei Argumente. Die Signatur kann man aus dem o.a. Ausschnitt aus der XML-Datei ablesen. Argument1: String, Argument2: String, Argument3: ComplexType
- Name des Interfaces: org.gtk.Private.UDisks2VolumeMonitor

24.9.5.1.1 Signatur des Signals 'VolumeAdded'

Interessant in Bezug auf die o.a. Aufgabenstellung ist folgender Abschnitt zum Signal mit dem Namen "VolumeAdded" aus dem sehr umfangreichen Inhalt des XML-Dokuments:

```
<signal name="VolumeAdded">
  <arg type="s" name="dbus_name"/>
  <arg type="s" name="id"/>
  <arg type="(sssssbbssa{ss}sa{sv})" name="volume"/>
</signal>
```

Folgendes kann festgehalten werden:

- Das Signal hat einen Namen, der in die richtige Richtung weist – VolumeAdded
- Das Signal hat genau drei Argumente.
- Zu jedem Argument wird ein (Daten-)Type angegeben.
- Jedes Argument besitzt einen Namen.

Um die Anzahl und Typen von Argumenten zu beschreiben, die von Methoden und Signalen benötigt werden, verwendet D-Bus eine String-Kodierung in Form von Signaturen.

- Die Argumente Arg1 und Arg2 haben den nativen Daten-Typ String.
- Das Argument 3 hat einen komplexen Datentyp: Ablesbar sind 6 aufeinander folgende Strings, denen 2 Wahrheitswerte folgen. Dann kommen zwei weitere Strings. Diesen Strings folgt eine Collection (Collection1 mit KeyTyp = String und DatenTyp = String). Der Collection1 folgt ein String, dem eine weitere Collection (Collection2 mit KeyTyp = String und DatenTyp = Variant) folgt.

Ein einfaches und ausdrucksstarkes Format! Interessant wird es nun für das Projekt DBusObserver1, wie man die Datentypen des 3. Arguments auch in Gambas auf einen komplexen Daten-Typ abbilden kann. Richtig – es gibt eine einfache Möglichkeit, indem Sie dafür den Daten-Typ *Struct* verwenden:

```
Public Struct StructVariant
  String1 As String
  String2 As String
  String3 As String
  String4 As String
  String5 As String
  String6 As String
  Boolean1 As Boolean
  Boolean2 As Boolean
  String7 As String
  String8 As String
  Collection1 As Collection
  String9 As String
  Collection2 As Collection
End Struct

Public structSignalArgument3 As New StructVariant
```

24.9.5.1.2 Projekt 1 – Teil 2

Der Quelltext für das Projekt ist überschaubar, wird deshalb vollständig angegeben und anschließend kommentiert:

```
[1]  ' Gambas class file
[2]
[3]  Public Struct StructVariant
[4]    String1 As String
[5]    String2 As String
[6]    String3 As String
[7]    String4 As String
[8]    String5 As String
[9]    String6 As String
[10]   Boolean1 As Boolean
[11]   Boolean2 As Boolean
[12]   String7 As String
[13]   String8 As String
[14]   Collection1 As Collection
[15]   String9 As String
[16]   Collection2 As Collection
[17] End Struct
[18]
[19] Public structSignalArgument3 As New StructVariant
[20] Public hDBusObserver As DBusObserver
[21]
[22] Private $cDBus As DBusConnection
[23] Private $iOSMessageType As Integer
[24] Private $sOSObjectPath As String
[25] Private $sOSMember As String
[26] Private $sOSInterface As String
[27] Private $sOSDestination As String
[28]
[29] Public Sub Form_Open()
[30]
[31]   FMain.Resizable = True
[32]
[33]   $cDBus = DBus.Session
[34]   $iOSMessageType = DBus.Signal
[35]   $sOSObjectPath = "/org/gtk/Private/RemoteVolumeMonitor"
[36]   $sOSMember = "VolumeAdded" ' Member -> DBus-SignalName
[37]   $sOSInterface = "org.gtk.Private.RemoteVolumeMonitor"
[38]   $sOSDestination = "*"
[39]
[40]   hDBusObserver = New DBusObserver($cDBus, $iOSMessageType, $sOSObjectPath, $sOSMember, $sOSInterface,
       $sOSDestination) As "hDBusObserver"
[41]
[42]   FMain.Caption = "DBus-Observer:  Signal" & " ==> " & $sOSMember
[43]
[44] End
[45]
[46] Public Sub hDBusObserver_Message()
[47]
[48]   Dim k As Integer = 1
[49]   Dim vElement As Variant
[50]   Dim aArguments As Variant[]
[51]   Dim sDestination As String
[52]   Dim aObserverMessageTypes As String[] = ["DBus.Method", "DBus.Reply", "DBus.Error", "DBus.Signal"]
[53]
[54]   aArguments = hDBusObserver.Message.Arguments
[55]
[56]   structSignalArgument3.String1 = aArguments[2][0]
[57]   structSignalArgument3.String2 = aArguments[2][1]
[58]   structSignalArgument3.String3 = aArguments[2][2]
[59]   structSignalArgument3.String4 = aArguments[2][3]
[60]   structSignalArgument3.String5 = aArguments[2][4]
[61]   structSignalArgument3.String6 = aArguments[2][5]
[62]   structSignalArgument3.Boolean1 = aArguments[2][6]
[63]   structSignalArgument3.Boolean2 = aArguments[2][7]
[64]   structSignalArgument3.String7 = aArguments[2][8]
[65]   structSignalArgument3.String8 = aArguments[2][9]
[66]   structSignalArgument3.Collection1 = aArguments[2][10]
[67]   structSignalArgument3.String9 = aArguments[2][11]
[68]   structSignalArgument3.Collection2 = aArguments[2][12]
[69]
[70]   If Not hDBusObserver.Message.Destination Then
[71]      sDestination = "All Applications"
[72]   Else
[73]      sDestination = hDBusObserver.Message.Destination
[74]   Endif
[75]
[76]   txaResults.Insert(gb.NewLine)
[77]   txaResults.Insert("Message-Destination = " & sDestination & gb.NewLine)
```

```
[78]   txaResults.Insert("Message-Interface = " & hDBusObserver.Message.Interface & gb.NewLine)
[79]   txaResults.Insert("Message-Member = " & hDBusObserver.Message.Member & gb.NewLine)
[80]   txaResults.Insert("Message-Object = " & hDBusObserver.Message.Object & gb.NewLine)
[81]   txaResults.Insert("Message-Sender (ID) = " & hDBusObserver.Message.Sender & gb.NewLine)
[82]   txaResults.Insert("Message-Number = " & hDBusObserver.Message.Serial & gb.NewLine)
[83]   txaResults.Insert("Message-Type = " & aObserverMessageTypes[hDBusObserver.Message.Type - 1])
[84]   txaResults.Insert(gb.NewLine & gb.NewLine)
[85]
[86]   txaResults.Insert("Arguments of the \"VolumeAdded\" Signal:" & gb.NewLine)
[87]   txaResults.Insert(String$(61, "-") & gb.NewLine)
[88]   txaResults.Insert("<signal name=\"VolumeAdded\"" & gb.NewLine)
[89]   txaResults.Insert("  <arg type=\"s\" name=\"dbus_name\"/>" & gb.NewLine)
[90]   txaResults.Insert("  <arg type=\"s\" name=\"id\"/>" & gb.NewLine)
[91]   txaResults.Insert("  <arg type=\"(sssssbbssa{ss}sa{sv})\" name=\"volume\"/>" & gb.NewLine)
[92]   txaResults.Insert("</signal>" & gb.NewLine & gb.NewLine)
[93]
[94]   txaResults.Insert("Number of arguments for Variant-Array 'Arguments' = " & aArguments.Count &
       gb.NewLine)
[95]   txaResults.Insert(gb.NewLine)
[96]   txaResults.Insert("String:" & gb.NewLine)
[97]   txaResults.Insert("Arguments[0] = " & aArguments[0] & gb.NewLine)
[98]   txaResults.Insert("String:" & gb.NewLine)
[99]   txaResults.Insert("Arguments[1] = " & aArguments[1] & gb.NewLine)
[100]
[101]    txaResults.Insert("Data-Type Struct:" & gb.NewLine)
[102]  ' 6x String
[103]    txaResults.Insert("Arguments[2][0] = " & aArguments[2][0] & gb.NewLine)
[104]    txaResults.Insert("Arguments[2][1] = " & aArguments[2][1] & gb.NewLine)
[105]    txaResults.Insert("Arguments[2][2] = " & aArguments[2][2] & gb.NewLine)
[106]    txaResults.Insert("Arguments[2][3] = " & aArguments[2][3] & gb.NewLine)
[107]    txaResults.Insert("Arguments[2][4] = " & aArguments[2][4] & gb.NewLine)
[108]    txaResults.Insert("Arguments[2][5] = " & aArguments[2][5] & gb.NewLine)
[109]  ' 2x Boolean
[110]    txaResults.Insert("Arguments[2][6] = " & aArguments[2][6] & gb.NewLine)
[111]    txaResults.Insert("Arguments[2][7] = " & aArguments[2][7] & gb.NewLine)
[112]  ' 2x String
[113]    txaResults.Insert("Arguments[2][8] = " & aArguments[2][8] & gb.NewLine)
[114]    txaResults.Insert("Arguments[2][9] = " & aArguments[2][9] & gb.NewLine)
[115]  ' Collection 1 -> KeyType = String and DataType = String
[116]    txaResults.Insert("Collection 1 = Arguments[2][10]" & gb.NewLine)
[117]    If structSignalArgument3.Collection1.Count > 0 Then
[118]       For Each vElement In structSignalArgument3.Collection1
[119]         txaResults.Insert("Element " & Str(k) & "  :   " & structSignalArgument3.Collection1.Key)
[120]         txaResults.Insert(" = " & vElement & gb.NewLine)
[121]         Inc k
[122]       Next
[123]     Else
[124]       txaResults.Insert("Attention: The collection 1 is empty!" & gb.NewLine)
[125]    Endif
[126]  ' 1x String
[127]    txaResults.Insert("Arguments[2][11] = " & aArguments[2][11] & gb.NewLine)
[128]    k = 1
[129]  ' Collection 2 -> KeyType = String and DataType = Variant
[130]    txaResults.Insert("Collection 2 = Arguments[2][12]" & gb.NewLine)
[131]    If structSignalArgument3.Collection2.Count > 0 Then
[132]       For Each vElement In structSignalArgument3.Collection2
[133]         txaResults.Insert("Element " & Str(k) & "  :   " & structSignalArgument3.Collection2.Key)
[134]         txaResults.Insert(" = " & vElement & gb.NewLine)
[135]         Inc k
[136]       Next
[137]    Else
[138]       txaResults.Insert("Attention: The collection 2 is empty!" & gb.NewLine)
[139]    Endif
[140]
[141]  End
[142]
[143]  Public Sub Form_Close()
[144]    hDBusObserver = Null
[145]    FMain.Close()
[146]  End
```

Kommentar:

- In den Zeilen 3 bis 27 werden die notwendigen Variablen deklariert.
- Die Belegung der Argumente für die Erzeugung eines DBusObserver-Objekts erfolgt in den Zeilen 33 bis 38.
- Sie können alternativ auch andere Signale abfangen (DriveConnected, DriveChanged, VolumeChanged, VolumeRemoved, MountAdded, DriveEject, MountPreUnmount, MountChanged). Dann müssen Sie jedoch für diese Signale deren Signaturen ermitteln, denn ein einfacher Austausch von $sOSMember = "VolumeAdded" gegen $sOSMember = "MountAdded" führt zu ei-

nem Fehler!

- Das Message-Ereignis in der Zeile 46 wird ausgelöst, wenn das zu beobachtende Signal auf dem D-Bus erkannt wird. Die komplette Nachricht wird in den Eigenschaften der virtuellen Klasse *DBusObserver.Message* gespeichert.
- In der Zeile 54 werden alle drei Argumente der Eigenschaft hDBusObserver.Message.Arguments in einem Variant-Array gespeichert.
- In den Zeilen 56 bis 68 werden die Elemente der definierten Struktur (structSignalArgument3) mit den Werten gefüllt, deren Daten-Typ vorher aus der Signatur für das dritte Argument abgelesen wurde.
- Die Werte aller Eigenschaften der Klasse *DBusObserver.Message* – jedoch ohne die Eigenschaft Arguments vom Daten-Typ Variant-Array werden in den Zeilen 77 bis 84 zugewiesen und in einer TextArea angezeigt.
- In den Zeilen 97 und 99 werden die Werte von Argument 1 und Argument 2 der Eigenschaft Arguments angezeigt. Danach erfolgt die Anzeige der Werte vom komplexen Argument 3 in den Zeilen ab 102.

Das Ergebnis der Beobachtung des Signals "VolumeAdded" des Objekts des Objekts org.gtk.Private.UDisks2VolumeMonitor kann sich sehen lassen:

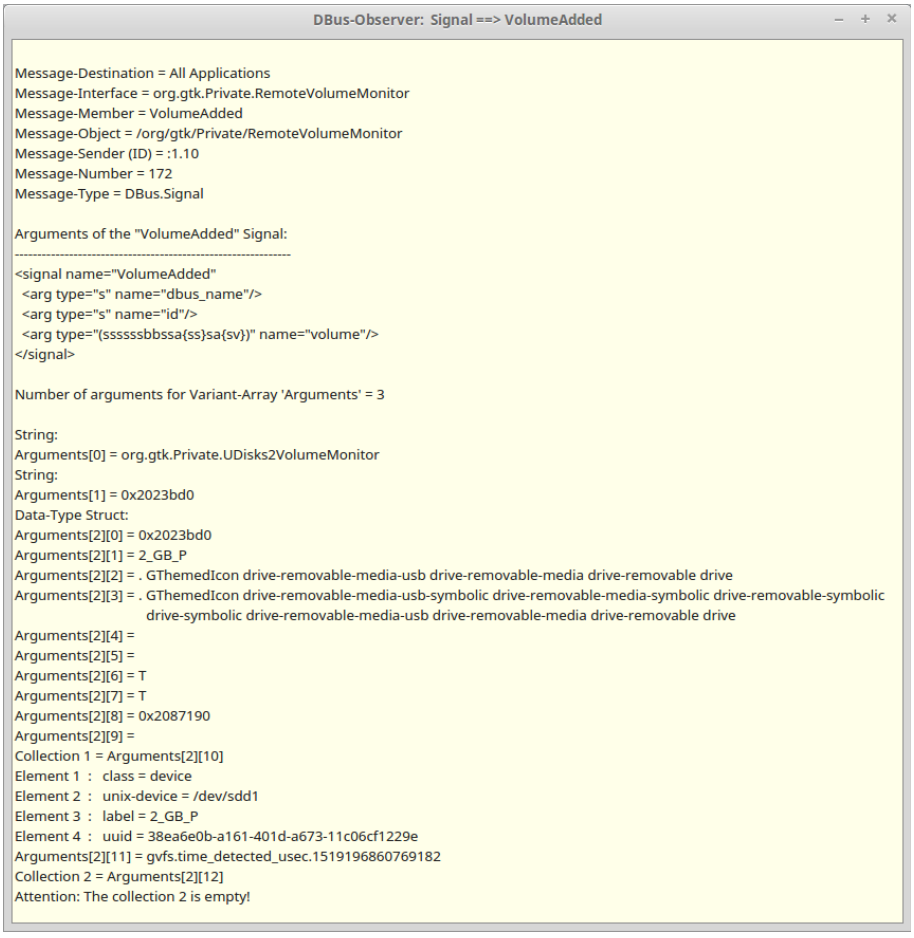

Abbildung 24.9.5.1.1: Auswertung der Beobachtung des Signals 'VolumeAdded' (Message-Member)

Das vorgestellte Projekt können Sie an andere zu beobachtende Nachrichten anpassen. Das gelingt Ihnen aber nur dann ohne Probleme, wenn Sie die Introspection für den ausgewählten Nachrichtentyp mit den empfohlenen Programmen mit Sorgfalt erneut ausführen, um u.a. die Signatur der Nachrichten zu ermitteln.

24.9.5.1.3 Projekt 2 – ObserverDBus – Signal 'MountAdded'

In einem weiteren Projekt wird Ihnen ein Observer für das Signal 'MountAdded' zur Erprobung zur Verfügung gestellt. Dabei wird auf den Daten-Typ Struct verzichtet und nur mit den Gambas-Datentypen

Integer, String, Array und Collection gearbeitet, wie ein Blick in die Definitionsliste zeigt:

```
Dim k As Integer = 1
Dim vElement As Variant
Dim cCollection As Collection
Dim aArray As String[]
Dim aArguments As Variant[]
Dim sDestination As String
Dim aObserverMessageTypes As String[] = ["DBus.Method", "DBus.Reply", "DBus.Error", "DBus.Signal"]
```

Die Signatur der Nachricht – gegenüber dem Projekt 1 – ist völlig anders:

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

Das ist die Ausgabe, wenn das Signal 'MountAdded' abgefangen und ausgewertet wurde:

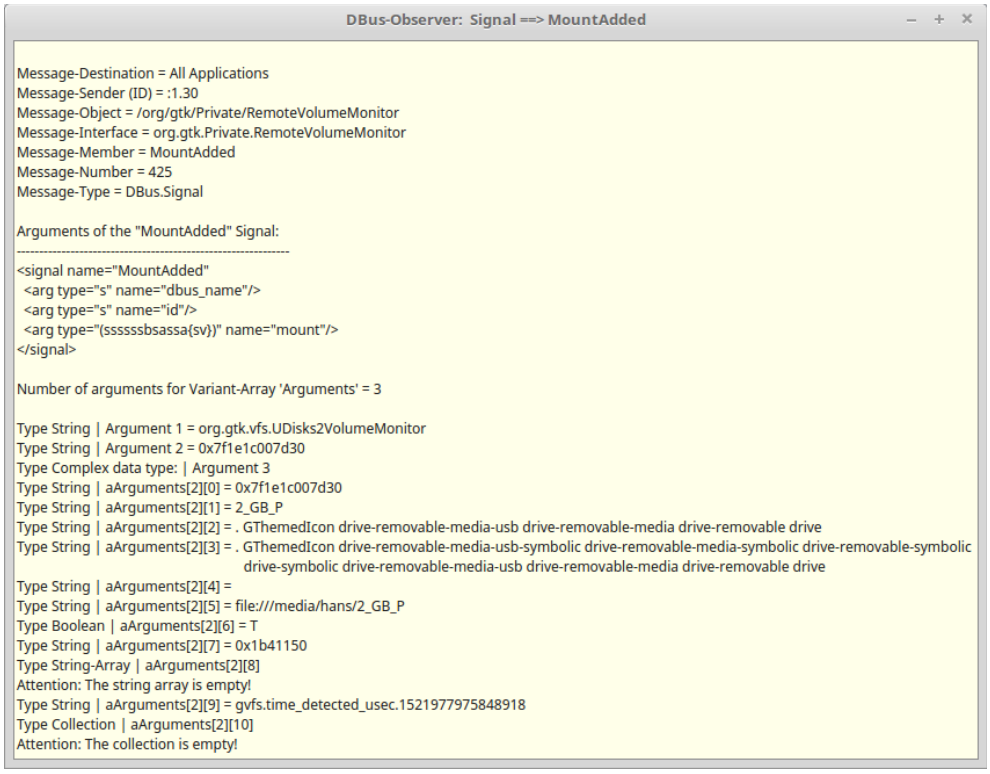

Abbildung 24.9.5.1.2: Auswertung der Beobachtung des Signals 'MountAdded'

Im Kapitel 24.9.8.0 werden Ihnen die beiden Klassen *DBusVariant* und *DBusValues* vorgestellt, mit denen Sie die passenden, dbus-gerechten Daten für *eigene* Nachrichten für selbst deklarierte Signaturen aus den Daten mit Gambas-Datentypen erzeugen können. Zwei Projekte, in denen Gambas-Programme (Server/Client) D-Bus-Objekte exportieren und Dienste anbieten oder nutzen, ergänzen die beschriebene Theorie.