20.11.0 Application

Diese Klasse *Application (gb)* gibt Auskunft über den aktuellen Prozess und das Gambas-Projekt, das in diesem Prozess läuft.

Die Beschreibung der Eigenschaft *Application.Args* unter Anwendungsaspekten finden Sie im Kapitel → 5.8.1.1 Klasse Args im Zusammenhang mit Optionen und Argumenten bei Gambas-Programmen. Die Klasse *Application (gb)* ergänzt die Möglichkeiten, System-Informationen auszulesen, auf die Sie über Eigenschaften der Klasse *System (gb)* oder der gleichnamigen Klasse *Application (gb.qt4)* zugreifen können.

20.11.0.1 Eigenschaften

Die Klasse *Application (gb)* verfügt u.a. über diese (statischen) Eigenschaften:

Eigenschaft	Datentyp	Beschreibung
Dir	Integer	Gibt das Arbeitsverzeichnis (CWD) beim Start der Anwendung zurück.
Handle	Integer	Gibt die System-PID (system process identifier) des Programms zurück.
Id	Integer	Synonym für Application.Handle
Name	String	Gibt den im Projekt-Eigenschaften-Dialog festgelegten Programm-Namen zurück. Der Name der Anwendung ist identisch mit dem Namen seines Projekt-Verzeichnisses.
Path	String	Gibt das Verzeichnis an, in dem das Projekt liegt.
Title	String	Gibt den Titel der Anwendung zur Laufzeit an wie er im Projekt-Eigenschaften-Dialog definiert wurde.
Version	String	Gibt die aktuelle Programm-Version an.
Priority	Integer	Setzt die Prozess-Priorität oder liest den Wert aus. Mit Application.Priority benutzt Gambas die POSIX-Funktionen getpriority() und setpriority(). Der Wertebereich geht von -20 (höchste Priorität) bis +19. Der Standardwert ist 0. Die Verringerung der Priorität ist ausdrücklich nur privilegierten Prozessen erlaubt.
Env	Env	Gibt eine virtuelle Collection von Strings zurück, welche die Prozess-Umgebungsvariablen enthalten. Die Klasse Env besitzt nur die Eigenschaft *Count* (Datentyp Integer).

Tabelle 20.11.0.1.1 : Wesentliche Eigenschaften der Klasse gb.application

20.11.0.2 Eigenschaft Application.Daemon

Setzen Sie die Eigenschaft *Application.Daemon* auf True, um den aktuellen Prozess zu einem Dämon zu machen. Um zu einem Dämon zu werden:

- teilt (gabelt) sich der der Prozess (erstellt eine Kopie) und beendet sich danach selbst. Die Kopie dagegen kann im Hintergrund als System-Dämon laufen,
- setzt er sein aktuelles Verzeichnis auf "/" und
- schließt die Standard-Eingabe, Standard-Ausgabe und die Standard-Fehlerausgabe.

Sobald Sie die Eigenschaft auf *True* gesetzt haben, können Sie das nicht mehr rückgängig machen und die Eigenschaft auf *False* setzen. Das temporäre Verzeichnis – in Gambas hat jeder Prozess ein Arbeitsverzeichnis in /tmp/gambas.<UID>/<PID>/ – wird umbenannt, da der *Process Identifier* (PID) sich beim Aufteilen ändert. Da ein Dämon im Hintergrund und periodisch arbeitet, während ein Prozess, der eine GUI hat, im Vordergrund bleiben soll und (benutzer-)interaktiv arbeitet, schließen sich beide Ansätze aus.

20.11.0.3 Spezial-Event-Handler

Die in den folgenden Absätzen beschriebenen globalen Event-Handler müssen in der Startklasse als statische Methoden definiert werden, um vom Interpreter berücksichtigt zu werden.

Application_Error()

Dieser Event-Handler wird aufgerufen, sobald ein Fehler im Gambas-Programm auftritt, der weder von einer CATCH-, TRY- noch FINALLY-Anweisung behandelt wurde. Sie kann von Nutzen sein, wenn ein Programm zum Beispiel eine genutzte Ressource in einen definierten Zustand bringen oder Daten retten muss, weil das Programm unerwartet abstürzt. Ein Beispiel dafür wäre die IDE, die mit dieser Methode alle offenen Dateien abspeichert, wenn sie abstürzt, um Datenverlust vorzubeugen.

Application_Error() ist eine Notfall-Routine. Der Interpreter ruft *Application_Error()* auf, kurz bevor er sich selbst aufgrund des Fehlers beendet. Als generelle Regel gilt: Sie sollten in *Application_Error()* niemals das Programm eigenhändig beenden. Diese Methode gehört zu dem Teil im Interpreter, der einen Programm-Absturz behandelt und muss als solcher auch agieren können. Es sind vom Programmierer nur die notwendigsten Maßnahmen zu ergreifen und die Methode sollte unbedingt ohne ME.Close oder QUIT zurückkehren. Für den Fall, dass in der Methode selbst ein Fehler auftritt, wird Application_Error() nicht erneut aufgerufen – das Programm wird beendet. Auf Informationen über den Fehler können Sie in Application_Error() nicht zugreifen.

Application_Read()

Es werden Daten gelesen, die an die *Standard-Eingabe* gesendet wurden. Wurde diese *statische* Methode in der Projekt-Startklasse definiert, dann wird die Standard-Eingabe durch den Interpreter beobachtet. Die Methode wird jedes Mal aufgerufen, wenn Daten über die Standard-Eingabe eingegeben wurden. Sie können die Daten auslesen, auswerten und dem Wert entsprechend reagieren.

20.11.0.4 Projekt

Das Projekt zeigt den Einsatz von *Application_Read()* als Spezial-Event-Handler. Der Internet-Radio-Client spielt nur den Lieblingssender des Autors – aber das mit Bravour. So viel Sound für so wenig Quelltext ist schon beachtlich:

```
' Gambas module file

Public mPlayer As New MediaPlayer

Public Sub Main()
 If mPlayer Then mPlayer = Null
 Start()
End ' Sub Main()

Public Sub Start()

  mPlayer = New MediaPlayer
  mPlayer.URL = "http://mp3channels.webradio.rockantenne.de/classic-perlen"
  mPlayer.Play()
  If mPlayer.Audio.Mute = True Then mPlayer.Audio.Mute = False
  FadeIn()

End ' Start()

Public Sub Application_Read() ' In einem Modul *ohne* das Schlüsselwort STATIC
 Dim sStandardInput As String

 Line Input #File.In, sStandardInput

 Select Case sStandardInput
   Case "q"
     FadeOut()
     mPlayer.Pause
     Print "Auf wieder-hören ... "
     Quit
   Case "p"
     mPlayer.Pause
   Case "r"
     mPlayer.Play
   Case "m"
     mPlayer.Audio.Mute = Not mPlayer.Audio.Mute
   Case "+"
     If mPlayer.Audio.Volume < 9.4 Then
        mPlayer.Audio.Volume += 0.5
     Endif
     Print "Volume = "; Round(mPlayer.Audio.Volume, -1)
   Case "-"
     If mPlayer.Audio.Volume > 0.6
```

```
        mPlayer.Audio.Volume -= 0.5
      Endif
      Print "Volume = "; Round(mPlayer.Audio.Volume, -1)
 End Select

End ' Application_Read()

Public Sub FadeIn()
  Dim fVolumeStart As Float = 2.0

  mPlayer.Audio.Volume = 0
  While mPlayer.Audio.Volume < fVolumeStart
    mPlayer.Audio.Volume += 0.05
    Print "*";
    Wait 0.2
  Wend
  Print
End ' FadeIn()

Public Sub FadeOut()
  While mPlayer.Audio.Volume > 0.2
    mPlayer.Audio.Volume -= 0.2
    Wait 0.1
  Wend
End ' FadeOut()
```

Das Kontrast-Programm bekommen Sie auf diesem Sender:

```
mPlayer.URL = "http://c22033l.i.core.cdn.streamfarm.net/22007mdrfigaro/live/3087mdr_figaro/live_de_128.mp3"
```

Den Client starten Sie in der Konsole alternativ so:

```
hans@linux:~/Rock4Me$ gbr3 ./r4m.gambas
hans@linux:~$ gbx3 $HOME/Rock4Me
```

- Mit den Tasten p, r, m steuern Sie die Funktionen *P*ause, Play (*R*un) und *M*ute (Stumm-Schaltung) des Clients.
- Die Prozeduren *FadeIn()* und *FadeOut()* sorgen jeweils für das Anheben und Absenken der Lautstärke beim Programm-Start und zum Programm-Ende.
- Die Lautstärke ändern Sie zur Laufzeit mit den Tasten - und +.
- Mit der Eingabe von q (Quit) beenden Sie das Konsolen-Programm.
- Jedem Zeichen folgt ein ENTER zur Aktivierung.