6.5 Klasse File

Diese Klasse wird verwendet für

- die Repräsentation einer mit dem OPEN()-Befehl geöffneten Datei,
- den Zugriff auf Standard-Eingangs-, Ausgangs- und Fehler-Streams,
- das Öffnen und Speichern einer Datei,
- die Manipulation von Dateipfaden mit den statischen Methoden der File-Klasse.

Sie können diese Klasse nicht direkt erzeugen. Sie müssen die Anweisung OPEN verwenden.

6.5.1 Eigenschaften

Die Klasse *File* verfügt über diese Eigenschaften:

Eigenschaft	Datentyp	Beschreibung
Blocking	Boolean	Gibt den Wahrheitswert True zurück oder setzt ihn, wenn der Stream blockiert wird. Wenn diese Eigenschaft auf True gesetzt ist, wird das Lesen aus dem Stream blockiert, wenn nichts zu lesen ist. Das Schreiben in den Stream wird blockiert, wenn beispielsweise der interne Systempuffer voll ist.
ByteOrder	Integer	Liefert die Byte-Reihenfolge oder setzt die Byte-Reihenfolge, die zum Lesen oder Schreiben von Binärdaten in den Stream verwendet wird. Die Eigenschaft kann folgende Werte annehmen: gb.BigEndian als "Big endian byte order" oder gb.LittleEndian als"Little endian byte order".
EndOfFile	Boolean	Diese Eigenschaft signalisiert, ob die letzte Verwendung von LINE INPUT das Ende der Datei erreicht hat, anstatt eine ganze Zeile mit einem Zeilenendezeichen zu lesen.
EndOfLine	Integer	Gibt das vom aktuellen Stream verwendete Zeilenumbruch-Trennzeichen zurück oder setzt es. Die möglichen Werte sind: gb.Unix für durch Chr$(10) getrennte Zeilen, gb.Windows für durch Chr$(13) und Chr$(10) getrennte Zeilen, gb.Mac für durch Chr$(13) getrennte Zeilen. Der Wert dieser Eigenschaft wird von LINE INPUT, PRINT und der Eigenschaft Stream.Lines verwendet. Beachten Sie, dass Sie mit gb.Unix sowohl Unix- als auch Windows-Zeilenformate lesen können. Aber es schreibt nur das Unix-Format!
Handle	Integer	Gibt den mit dem aktuellen Stream verknüpften Systemdatei-Deskriptor zurück.
IsTerm	Boolean	Gibt True zurück, wenn ein Stream einem Terminal zugeordnet ist.
Lines	.Stream.Lines	Gibt ein virtuelles Objekt (Typ String-Array) zurück, mit dem Sie einen Stream zeilenweise auslesen können.
Tag	Variant	Gibt den dem Stream zugeordneten Wert der Tag-Eigenschaft zurück oder setzt den Wert. Diese Eigenschaft vom Datentyp Variant ist für die freie Verwendung durch den Programmierer bestimmt und wird nicht von der Komponente verwendet.
Term	.Stream.Term	Gibt ein virtuelles Objekt zurück, mit dem das dem Stream zugeordnete Terminal verwaltet werden kann. Die virtuelle Klasse Stream.Term (gb) hat die Eigenschaften Echo, FlowControl, Height, Width und Name sowie die Methode Resize. Link: http://gambaswiki.org/wiki/comp/gb/.stream.term

Tabelle 6.5.1.1 : Eigenschaften der Klasse File

Beispiel für die Verwendung der Eigenschaft *Lines*:

```
Dim hStream As Stream
Dim aString As String
...
For Each aString In hStream.Lines
  ...
Next
```

6.5.2 Statische Eigenschaften

Die Klasse File verfügt über diese drei statischen Eigenschaften:

Eigenschaft	Datentyp	Beschreibung
In	File	Liefert den Standard-Eingabestrom.
Out	File	Liefert den Standard-Ausgabestrom.
Err	File	Liefert den Standard-Fehlerausgabestrom.

Tabelle 6.5.2.1 : Statische Eigenschaften der Klasse File

Passende Informationen erhalten Sie auf der Seite: https://wiki.ubuntuusers.de/Shell/Umleitungen zum folgenden Beispiel. Es zeigt, wie man auf der Standard-Eingabe, auf der Standard-Ausgabe und auf der Standard-Fehlerausgabe liest und schreibt und präsentiert die Verwendung der statischen Eigenschaften File.In, File.Out und File.Err. Jede Zeile, die auf der Standard-Eingabe geschrieben wird, wird zuerst auf der Standard-Ausgabe und dann auf der Standard-Fehlerausgabe zurückgegeben.

```
Public Sub Main()
  Dim inputLine As String
  ' Loop until the end of the standard input file stream
  While Not Eof(File.In)
    ' Read a line from standard input
    Line Input #File.In, inputLine ' Read data
    ' Print to standard output
    Print #File.Out, inputLine ' Write data
    ' Print to standard error
    Print #File.Err, inputLine ' Write data
  Wend
End
```

Erzeugen Sie ein Projekt (Kommandozeilen-Anwendung ioe (input output error)) und platzieren Sie den obigen Code in der Datei Main.module. Wenn Sie dieses Programm in der Gambas-IDE ausführen, scheint die Anwendung zu hängen. Wenn Sie einen Text im Konsolenfenster der IDE eingeben und mit Enter abschließen, so wird jede Zeile zweimal in der Konsole zurückgegeben. Da die Anwendung jedoch nie das Ende der Datei für den Standard-Eingabestrom sieht, müssen Sie das Projekt stoppen. Um eine bessere Vorstellung davon zu bekommen, was das Beispiel macht, erzeugen Sie eine ausführbare Gambas-Datei und nennen diese ioe.gambas. Öffnen Sie ein Terminal und wechseln Sie in das Verzeichnis, in dem Sie die ausführbare Datei gespeichert haben.

Wenn Sie den folgenden Befehl eingeben:

```
ls -a | ./ioe.gambas
```

so werden Sie jede Ausgabe-Zeile des Befehls ls -a sehen, jedoch stets zweimal. Einmal von der Standard-Ausgabe (stdout) und einmal von der Standardfehler-Ausgabe (stderr). Die Anwendung sieht jetzt aber das Ende des Standard-Eingabestroms und endet somit korrekt. Mit dem Befehl

```
ls -a | ./ioe.gambas > files.txt
```

leiten Sie einerseits den Standard-Ausgabestrom des Befehls ls -a in eine Datei um und Sie sehen andererseits den Standardfehler-Ausgabestrom im Terminalfenster.

6.5.3 Methoden

Die Klasse File verfügt über diese sechs Methoden:

Methode	Rückgabetyp	Beschreibung
Begin()	-	Startet das Puffern der in den Stream geschriebenen Daten, so dass beim Aufruf der Send-Methode alles auf einmal gesendet wird.
Close()	-	Schließt den Stream. Die Methode entspricht exakt der CLOSE-Anweisung.
Drop()	-	Löscht die Daten, die seit dem letzten Aufruf der Methode Begin() gepuffert wurden.
ReadLine([Escape As String])	String	Liest eine Textzeile aus dem Stream, genau so wie die Anweisung LINE INPUT. Wenn Escape angegeben ist, werden Zeilenumbrü-

Methode	Rückgabetyp	Beschreibung
		che zwischen zwei Escape-Zeichen ignoriert. Diese Methode ist sehr nützlich beim Lesen von CSV-Dateien.
Send()	-	Sendet alle Daten auf einmal, die seit dem letzten Anruf der Methode Begin() gepuffert wurden.
Watch (Mode As Integer, Watch As Boolean)	-	Startet oder stoppt die Beobachtung des Stream-Dateideskriptors zum Lesen oder Schreiben, nachdem er geöffnet wurde. Modus ist der Beobachtungstyp: gb.Read zur Beobachtung beim Lesen oder gb.Write zur Beobachtung beim Schreiben. Watch ist TRUE, um die Beobachtung zu aktivieren und FALSE, um sie zu deaktivieren.

Tabelle 6.5.3.1 : Methoden der Klasse File

6.5.4 Ereignisse

Die Klasse besitzt u.a. diese zwei ausgewählten Ereignisse:

Ereignis	Beschreibung
Read()	Dieses Ereignis wird ausgelöst, wenn etwas aus der Datei zu lesen ist. Die Datei muss zu diesem Zweck mit dem Schlüsselwort WATCH geöffnet worden sein. Siehe OPEN für weitere Details.
Write()	Dieses Ereignis wird ausgelöst, wenn das Schreiben in die Datei möglich ist. Die Datei muss mit dem Schlüsselwort WATCH geöffnet worden sein. Siehe OPEN für weitere Details.

Tabelle 6.5.4.1 : Ereignisse der Klasse File

6.5.5 Statische Methoden

Von besonderem Interesse dürften die folgenden (statischen) Methoden sein – vor Allem die Load()- und Save()-Methode:

Methode	Beschreibung
BaseName(Path As String)	Gibt den Namen einer Datei ohne Erweiterung zurück.
SetBaseName(Path As String, NewBaseName As String)	Ersetzt nur den Basisnamen eines Pfades und gibt den geänderten Pfad zurück.
Dir(Path As String)	Gibt den Verzeichnisteil eines Dateipfades zurück.
SetDir(Path As String, NewDir As String)	Ersetzt nur den Verzeichnisteil eines Pfades und gibt den geänderten Pfad zurück.
Ext(Path As String)	Gibt die Erweiterung (ohne Punkt) eines Dateinamens zurück.
SetExt(Path As String, NewExt As String)	Ersetzt nur die Dateierweiterung eines Pfades und gibt den geänderten Pfad zurück.
Name(Path As String)	Gibt den kompletten Dateinamen mit Extension eines Dateipfades zurück.
SetName(Path As String, NewName As String)	Ersetzt den Dateinamen eines Pfades und gibt den geänderten Pfad zurück.
Load(FilePath As String)	Lädt eine Datei und gibt ihren Inhalt als String zurück.
Save(FilePath As String, Data As String)	Speichert den Inhalt einer Zeichenkette (Data) in eine Datei mit dem Pfad *FilePath*.
IsHidden (Path As String)	Gibt True zurück, wenn ein Dateipfad versteckt ist. Path ist der zu prüfende Dateipfad. Ein Dateipfad wird ausgeblendet, wenn er selbst oder eines seiner übergeordneten Verzeichnisse ausgeblendet ist. Unter UNIX sind versteckte Dateien oder Verzeichnisse diejenigen, deren Name mit einem Punkt beginnt.
IsRelative (Path As String)	Gibt True zurück, wenn Path ein relativer Pfad ist. Ein relativer Pfad ist ein Pfad, der nicht mit dem Zeichen / oder ~ beginnt.

Tabelle 6.5.5.1 : Statische Methoden der Klasse File

6.5.6 Beispiele – Einsatz von ausgewählten Methoden der Klasse File

6.5.6.1 Beispiel 1 – Anzeige von Dateipfad-Eigenschaften

```
Public Sub btnDisplayProperties_Click()
  If Dialog.OpenFile() Then Return
  Print "Datei-Pfad: " & Dialog.Path
  Print "Verzeichnis der Datei: " & File.Dir(Dialog.Path)
  Print "Dateiname (mit Extension): " & File.Name(Dialog.Path)
  Print "Dateiname (ohne Extension): " & File.BaseName(Dialog.Path)
  Print "Datei-Extension (ohne Punkt): " & File.Ext(Dialog.Path)
End
```

6.5.6.2 Beispiel 2 – File.Load(Path) und File.Save(Path)

In einem Datei-Öffnen- oder Datei-Speichern-Dialog können Sie Datei-Filter einsetzen. Mit regulären Ausdrücken können Sie jedoch die Filter wesentlich differenzierter setzen:

```
Dialog.Title = "Importieren Sie eine Text-Datei!"
Dialog.Filter = ["tb*.txt", "Text-Dateien"] ' Dateiname beginnt mit tb
Dialog.Filter = ["[0-9]*.txt", "Text-Dateien"] ' Dateiname beginnt mit einer Ziffer
Dialog.Filter = ["tb[1-3]*.txt; ta[1-3]*.log", "Text-Dateien"]
```

Mit dem letzten Filter werden alle Dateien im aktuellen Verzeichnis selektiert, die mit der Buchstabenfolge 'tb' beginnen, der *eine* Ziffer aus dem Bereich von 1-3 folgt und dann eine Folge von beliebigen Zeichen. Die Extension ist entweder .txt oder .log.Dabei wird zum Beispiel für Text-Dateien oft ein Filter der folgenden Art verwendet:

```
Private $sCurrentFilePath As String

Public Sub btnFileLoad_Click()

  Dialog.Filter = ["*.txt;*.log;*.xml;*.conf", "Text files"]
  If Dialog.OpenFile() Then Return
  $sCurrentFilePath =  Dialog.Path

  txaTextArea.Text = File.Load(Dialog.Path)
  Catch
    Message.Info(Error.Text)

End
```

```
Public Sub btnFileSave_Click()

  If Dialog.SaveFile() Then Return

  File.Save(Dialog.Path, txaTextArea.Text)
  Catch
    Message.Info(Error.Text)

End
```

6.5.6.3 Beispiel 3 – Einsatz ausgewählter statischer Methoden der Klasse File

```
Dim filePath As String

Print "* A standard type of path"
  filePath = "/my/path/file.ext"
Print filePath
Print File.SetBaseName(filePath, "new-name")

Print "\\n* Try a path with two extensions"
  filePath = "/my/path/file.ext1.ext2"
Print filePath
Print File.SetBaseName(filePath, "new-name")

Print "\\n* A path with just an extension"
  filePath = ".ext"
Print filePath
Print File.SetBaseName(filePath, "new-name")

Print "\\n* A path without a file name"
  filePath = "/my/path/.ext"
Print filePath
Print File.SetBaseName(filePath, "new-name")

Print "\\n* A path without a file name or extension"
  filePath = "/my/path/"
```

```
Print filePath
Print File.SetBaseName(filePath, "new-name")
```

Ausgaben in der Konsole der IDE:

```
* A standard type of path
/my/path/file.ext
/my/path/new-name.ext

* Try a path with two extensions
/my/path/file.ext1.ext2
/my/path/new-name.ext2

* A path with just an extension
.ext
new-name.ext

* A path without a file name
/my/path/.ext
/my/path/new-name.ext

* A path without a file name or extension
/my/path/
/my/path/new-name
```

6.5.6.4 Projekt – Steuerung eines Players

Mit folgendem Quelltext geben Sie Text mit *File.Out* in der Konsole aus und steuern den Musik-Player über die Standard-Eingabe *File.In*.

So starten Sie den Player in einer Konsole aus dem Projekt-Verzeichnis heraus:

```
hans@mint-183 ~/GB3BUCH/6K_Stream/.../MusicPlayer $ gbr3 ./music_player_console.gambas
```

Quelltext:

```
' Gambas module file

' Über die Standard-Eingabe (Terminal) werden Zeichen eingelesen, um damit den MediaPlayer zu steuern
' Hinweise: http://www.mplayerhq.hu/DOCS/man/de/mplayer.1.html

Public mPlayer As New MediaPlayer

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

Public Sub Start()

  mPlayer = New MediaPlayer
  mPlayer.URL = "http://mp3channels.webradio.rockantenne.de/classic-perlen"
  mPlayer.Audio.Volume = +1.0
  mPlayer.Play()

  Print #File.Out, ""
  Print #File.Out, "--------------------------------------"
  Print #File.Out, "Instructions for use"
  Print #File.Out, "--------------------------------------"
  Print #File.Out, "+  ► Audio.Volume ▲"
  Print #File.Out, "-  ► Audio.Volume ▼"
  Print #File.Out, "p  ► Player.Pause"
  Print #File.Out, "r  ► Player.Run (After a pause)"
  Print #File.Out, "m  ► Audio.Mute (off/on)"
  Print #File.Out, "q  ► Player.Stop"
  Print #File.Out, "--------------------------------------"
  Print #File.Out, "Each command is followed by <ENTER>."
  Print #File.Out, "--------------------------------------"
  Print #File.Out, ""

End

Public Sub Application_Read()

  Dim sInput As String
  Dim fDeltaVolume As Float

  If mPlayer.Audio.Volume > 1.1 Then
    fDeltaVolume = 1.0
  Else
```

```
    fDeltaVolume = 0.1
  EndIf

  Line Input #File.In, sInput

  Select Case sInput
    Case "q"
      mPlayer.Stop()
      Quit
    Case "p"
      mPlayer.Pause()
    Case "r" ' Run
      mPlayer.Play()
    Case "m" ' Toggle switch: mute on/mute off
      mPlayer.Audio.Mute = Not mPlayer.Audio.Mute
    Case "+"
      If mPlayer.Audio.Volume > 0.09 And mPlayer.Audio.Volume < 9.0 Then
         mPlayer.Audio.Volume += fDeltaVolume
      Endif
    Case "-"
      If mPlayer.Audio.Volume > 0.2 And mPlayer.Audio.Volume < 10.0 Then
         mPlayer.Audio.Volume -= fDeltaVolume
      Endif
  End Select

End
```

Kommentar:

- Zuerst wird geprüft, ob bereits eine Instanz des Players existiert. Trifft das zu, wird das existierende Player-Objekt zerstört und dann ein neuer Player gestartet.
- Nach dem Start mit einer vorgegebenen Lautstärke wird mit 'Print #File.Out, TestString' eine Anleitung zeilenweise in der Konsole ausgegeben.
- Danach können Sie den Player durch die Eingabe von +, -, p, r, m und q (mit jeweils abschließendem Enter) steuern.
- Die Eingaben werden von der Standard-Eingabe mit 'Line Input #File.In, sInput' gelesen und in einer Select-Case-Kontrollstruktur ausgewertet.