6.0 Klasse Stream (gb)

Ein Stream ist als Datenstrom aufzufassen, der zwischen Prozessen oder einem Prozess und einer Datenendverbindung fließt.

Übersicht zu den unterschiedlichen Arten von Datenendverbindungen, die in Linux global als Datei bezeichnet werden:

- Reguläre Dateien (regular file)
- Verzeichnisse (directory)
- Symbolische Links (symbolic link)
- Blockorientierte Geräte (block device)
- Zeichenorientierte Geräte (char device)
- Datenverbindungen zwischen Prozessen nach dem FIFO-Prinzip (named pipe)
- Kommunikationsendpunkte (unix-socket)

Hinweise:

- Zu den regulären Dateien zählen die Dateien, die Sie zum Beispiel als Bild- oder Text-Dateien kennen.
- Verzeichnisse sind als Container für Dateien aufzufassen.
- Unter einem symbolischen Link können Sie sich eine Verknüpfung zu einer Datei oder einem Verzeichnis vorstellen.
- Bei den Geräte-Dateien wird zwischen den blockorientierten Geräten wie RAM, Festplatten, Partitionen oder USB-Stick und den zeichen-orientierten Geräten wie serielle und parallele Schnittstelle, Terminal oder Sound-Schnittstelle unterschieden. Unter Linux entspricht die Gerätedatei /dev/stdin (Datei-Deskriptor = 0) der Standard-Eingabe, die Gerätedatei /dev/stdout (Datei-Deskriptor = 1) der Standard-Ausgabe und die Gerätedatei /dev/stderr (Datei-Deskriptor = 2) der Standard-Fehlerausgabe. Die Standard-Fehlerausgabe ist ein weiterer Ausgabe-Datenstrom, um Fehler- und Statusmeldungen auszugeben. Unter Gambas kann auf die Gerätedateien (stdin, stdout und stderr) über die in der Klasse File bereitgestellten statischen Eigenschaften File.In, File.Out und File.Err zugegriffen werden. Im Kapitel "6.5 Klasse File" finden Sie ein Beispiel, wie Sie Daten in File.In schreiben und von File.Out oder File.Err auslesen.
- Eine Named Pipe ist ein flüchtiger Speicher, in den ein Prozess Daten schreibt und ein anderer aus der Named Pipe die Daten ausliest. Als flüchtiger FIFO-Speicher ist er deshalb nach dem Auslesen von Daten leer! Eine Named Pipe repräsentiert eine Simplex-Verbindung. Sie können auf einer gegebenen Pipe entweder nur lesen oder schreiben.
- Sockets verkörpern den Stream, der durch sie verschickt wird. Ist ein Socket mit einem anderen verbunden, repräsentiert er diesen Datenstrom. Symbolisiert ein Socket einen Datenstrom aus einer Verbindung ins Internet, dann nutzen Sie einen TCP-Socket, für den Sie einen Hostnamen und einen Port angegen müssen. Im Gegensatz dazu müssen Sie bei einem Unix-Socket als lokalem Socket nur einen Pfad angeben – zum Unix-Socket gehört dann eine spezielle Datei im Dateisystem. Mit dem Kommando *~ $ netstat -a -p --unix* können Sie die aktuellen lokalen Unix-Sockets in einer Konsole auflisten – Sie werden überrascht sein!

Die Schnittstelle Stream gibt an, welche Methoden implementiert sind. Ein Stream verfügt u.a. über diese vier Methoden:

- Öffnen (open),
- Daten lesen (read),
- Daten schreiben (write),
- Schließen(close).

Die Klasse Stream ist die übergeordnete Klasse für jedes Gambas-Objekt, das einen Stream (Datenstrom) repräsentiert. Das bedeutet, das jede Klasse, die von der Klasse Stream (gb) als Basis-Klasse erbt, mindestens über diese vier Methoden verfügt. Diese (Basis-)Klasse können Sie nicht erzeugen. Welche Funktionalität eines Stream-Objekts Sie verwenden können, hängt von seiner genauen Klasse und auch von der Art der Objekterzeugung ab.

Für die praktische Arbeit ist es somit weitgehend unerheblich, ob Sie Daten zum Beispiel in eine reguläre Datei, in eine serielle Schnittstelle oder in eine Unix-Socket schreiben oder aus diesen Daten lesen.

Hier finden Sie die Liste der Klassen, die von der Klasse Stream erben und im Gambas-Buch beschrieben werden:

- Compress Kapitel 28.1.1
- Uncompress Kapitel 28.1.2
- Process Kapitel 21
- SerialPort Kapitel 24.1.5
- Curl Kapitel 24 (FTPClient und HTTPClient.)
- File Kapitel 6.5
- Named Pipe Kapitel 6.2.1
- Socket Kapitel 24 (TCP- und Local-Unix-Socket)
- VideoDevice Kapitel 24

6.0.1 Eigenschaften

In Gambas ist ein Stream ein Objekt, das von der Stream-Klasse (gb) erbt und daher deren Eigenschaften und Methoden implementiert. Die Klasse Stream 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 6.0.1.1 : Eigenschaften der Klasse Stream

6.0.2 Methoden

Die Klasse Stream besitzt 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.

Methode	Rückgabetyp	Beschreibung
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 Zeilenumbrüche 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 6.0.2.1 : Methoden der Klasse Stream

6.0.3 Exkurs

6.0.3.1 Projekt

Das Projekt im Downloadbereich demonstriert, wie Sie mit Hilfe der Type-Eigenschaft eines Stat-Objektes die Art einer Datei ermitteln:

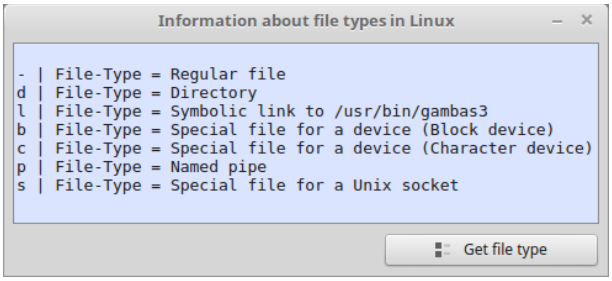

Abbildung 6.0.3.1.1: Ermittlung der Art von Dateien

Hinweis:

Die Zeichen -, d, l, b, c, p und s charakterisieren die Art einer Datei im Zusammenhang mit dem Befehl ls -l und der Anzeige der Ausgabe in einer Konsole.

6.0.3.2 Einsatz file-Befehl

Auch in einer Konsole können Sie mit dem file-Befehl schnell feststellen, um welche Art es sich bei einer vorgegebenen Datei handelt:

```
Reguläre Dateien (regular file)
-------------------------------
hans@mint-183 ~ $ file nwm.xml
nwm.xml: exported SGML document, ASCII text

Verzeichnisse (directory)
-------------------------
hans@mint-183 ~/BildTon $ file Fractals
Fractals: directory

Symbolische Links (symbolic link)
---------------------------------
hans@mint-183 ~/Schreibtisch $ file Formatting_DokuWiki_Tables
Formatting_DokuWiki_Tables: symbolic link to /home/hans/DW/0_DW_Convert/librewriter2dokuwiki.gambas

Blockorientierte Geräte (block device)
--------------------------------------
hans@mint-183 ~/Schreibtisch $ file /dev/sdd1
```

```
/dev/sdd1: block special (8/49)

Zeichenorientierte Geräte (char device)
---------------------------------------
hans@mint-183 ~ $ file /dev/ttyUSB0
/dev/ttyUSB0: character special (188/0)

Datenverbindungen zwischen Prozessen (named pipe)
-------------------------------------------------
hans@mint-183 ~ $ mknod mypipe p  ' Named Pipe erzeugen
hans@mint-183 ~ $ file mypipe
mypipe: fifo (named pipe)
hans@mint-183 ~ $ rm mypipe       ' Named Pipe löschen

Kommunikationsendpunkt (unix socket)
------------------------------------
hans@mint-183 ~ $ file /run/user/1000/unix_socket.sock
/run/user/1000/unix_socket.sock: socket
```

6.0.3.3 Das #-Zeichen

In der Dokumentation steht zum Bespiel

```
FLUSH [ [ # ] Stream ]
```

wobei das #-Zeichen als optional gekennzeichnet ist. In Gambas können Sie das #-Zeichen dem Variablennamen eines Stream-Objektes voranstellen, aber nur für die Instruktionen CLOSE, READ, WRITE, SEEK, INPUT, LINE INPUT, PRINT, UNLOCK und FLUSH. Das geht nicht in Funktionsaufrufen wie LOF(…), EOF(...) oder der Instruktion LOCK.

In einem Socket_Read-Ereignis können Sie zum Beispiel folgende Zuweisung verwenden:

```
vData = Read #Last, -4096
```

In Last wird das Socket-Objekt zurückgegeben, das das Read-Event ausgelöst hat und von dem Daten gelesen werden können. Das "#Last" sollten Sie dem "Last" vorziehen, weil man "Last" allein nicht ansieht, dass es sich in diesem Fall um einen Stream handelt.