NAME

Lua-AKFAvatar Referenz − AKFAvatar spezifische Funktionen für Lua

BESCHREIBUNG

Dies ist eine Referenz für die Lua-Anbindung für AKFAvatar. Sie beschreibt nicht die Sprache Lua. Eine Referenz-Anleitung für die Sprache Lua findet man im Internet unter **http://www.lua.org/manual/5.2/**, oder auf Deutsch unter **http://lua.coders-online.net/**.

Keine Angst. Man muss nicht alles gleich verstehen oder gar lernen. Man kann schon eine ganze Menge mit nur ganz wenigen Befehlen anfangen. Einfach mal die Teile heraussuchen, die einen interessieren und damit herumexperimentieren. Es gibt viele Aspekte von Lua oder AKFAvatar, die man wahrscheinlich nie benötigt.

Datei-Formate

Für Bilder werden die Formate **XPM**, **XBM** und unkomprimiertes **BMP** auf jeden Fall unterstützt. Weitere Formate werden unterstützt, wenn **SDL_image** und weitere Bibliotheken installiert sind.

Für Audio werden die Formate **AU** und **WAV** unterstützt. Dateien können als PCM, µ-law oder A-law kodiert sein. Es gibt auch ein Lua-Modul namens **akfavatar-vorbis**, welches Unterstützung für Ogg Vorbis Audio-Dateien hinzufügt.

Grundfunktionalität

local avt = require "lua-akfavatar"

Bevor man die AKFAvatar-Anbindung verwenden kann, muss man sie mit dieser Anweisung erstmal laden. Über die Tabelle *avt* kann man dann die folgenden Funktionen abrufen.

avt.encoding(*encoding***)**

Legt die Text-Kodierung fest. Wenn diese Funktion nicht verwendet wird, wird nur "ASCII" unterstützt.

Unterstützte Kodierungen:
ASCII, UTF-8,
ISO-8859-1, -2, -3, -4, -5, -7, -8, -9, -10, -11, -13, -14, -15, -16,
KOI8-R, KOI8-U,
IBM437 (PC-8), IBM850 (DOS-Latin-1),
WINDOWS-1250, WINDOWS-1251, WINDOWS-1252
SYSTEM

avt.get_encoding()

Fragt die Text-Kodierung ab.

avt.title(*Titel [,Kurzname]***)**
avt.set_title(*Titel [,Kurzname]***)**

Legt den Titel und den Kurznamen fest.

avt.start(*Modus***)**

Diese Funktion öffnet das grafische Fenster oder wechselt in den Grafikmodus. Falls es schon gestartet war, wird vieles zurück gesetzt.

Modus ist entweder "auto", "window", "fullscreen", oder "fullscreen no switch". Die Vorgabe ist "auto", die einen brauchbaren Modus verwendet, oder die Einstellung unverändert lässt, wenn es bereits gestartet war.

Wenn man die Kodierung, den Titel oder die Hintergrund-Farbe setzen will, sollte man es machen, bevor man **avt.start()** aufruft. Alle anderen Funktionen sollten erst danach verwendet werden.

Beispiel:

```
local avt = require "lua-akfavatar"
avt.encoding("ISO-8859-1")
avt.title("Mein Programm")
avt.set_background_color("sky blue")
avt.start()
```

```
avt.start_audio()
avt.avatar_image("default")
avt.tell("Hallo Welt!")
avt.wait_button()
```

Viele der folgenden Funktionen rufen automatisch **avt.start()** auf, wenn es noch nicht gestartet ist.

avt.started()
Gibt *true* zurück, wenn AKFAvatar bereits gestartet ist, ansonsten gibt es *false* zurück.

avt.avatar_image(*Daten***)**
Lädt das Avatar-Bild von den *Daten*. Die *Daten* können entweder "default" oder "none" sein, oder ein String mit Bilddaten, oder eine Tabelle mit Strings von XPM-Daten.

avt.avatar_image_file(*Dateiname***)**
Lädt das Avatar-Bild aus einer Datei. Die Strings "default" oder "none" werden auch akzeptiert.

avt.set_avatar_name(*[Name]***)**
Setzt den Namen für den Avatar. Dies muss nach dem Setzen des Bildes geschehen.

avt.set_avatar_mode(*Modus***)**
Legt den Avatar-Modus fest. Der *Modus* ist entweder "say" (sagen), "think" (denken), "header" (Überschrift) oder "footer" (Fußzeile). Der Avatar-Modus wird nur angewandt, wenn ein Avatar-Bild vorhanden ist.

avt.say(...**)**
avt.write(...**)**
Schreibt Text in der Sprechblase.

Man kann Strings oder Zahlen verwenden. Es funktioniert so ähnlich wie **io.write()**, aber es schreibt halt in die Sprechblase anstatt auf die Standardausgabe.

Beispiel:
```
avt.say("Drei Äpfel kosten ", 3 * Apfelpreis, " Euro.\n").
```

avt.print(...**)**
Schreibt Text in der Sprechblase. Es funktioniert so ähnlich wie **print**, aber es schreibt halt in die Sprechblase anstatt auf die Standardausgabe.

Der Unterschied zu **avt.say()** ist, dass die Werte in der Ausgabe durch Tabulatoren getrennt werden und jeder Wert automatisch durch **tostring()** umgewandelt wird. Außerdem wird die Zeile durch einen Zeilenumbruch abgeschlossen.

Mann kann einfach den print-Befehl mit diesem Befehl ersetzen: **print = avt.print**. (Das Skript ‘interactive_lua.lua’ macht das so.)

Die Funktion **avt.say()** eignet sich besser für Programme, während diese Funktion besser für die interaktive Verwendung und für’s Debuggen geeignet ist.

avt.tell(...**)**
Schreibt Text in der Sprechblase, aber zuvor wird die Größe der Sprechblase angepasst, so dass der Text exakt reinpasst.

Beispiel:
```
avt.tell("Drei Äpfel kosten ", 3 * Apfelpreis, " Euro.").
```

Achtung: Diese Funktion hat noch immer Probleme mit Tabulatoren ("\t").

avt.newline()
Beginne eine neue Zeile. Das selbe, wie "\n" in **avt.say()**, aber schneller.

avt.set_text_delay(*[Verzögerung]***)**
Aktiviere den Langsamschreibmodus. Wenn *Verzögerung* nicht angegeben ist, wird ein Vorgabewert verwendet. Um den Langsamschreibmodus zu daktivieren, kann man den Wert 0 für die *Verzögerung* verwenden.

avt.clear()
Löscht das Text-Feld oder den Bereich. Wenn noch keine Sprechblase angezeigt wird, wird diese hierdurch gezeichnet.

avt.flip_page()
Wartet eine Weile und löscht dann das Text-Feld. Dasselbe wird durch "\f" in **avt.say()** erreicht. Siehe auch **avt.set_flip_delay(delay)**.

avt.move_in()
Bewegt den Avatar herein.

avt.move_out()
Bewegt den Avatar heraus.

avt.pager(*Text [,Anfangszeile]***)**
Zeigt einen längeren Text in einer Textbetrachter-Anwendung.

Wenn die *Anfangszeile* angegeben und größer als 1 ist, dann fängt er in der Zeile an. Man kann von da aus aber immer noch zurück scrollen.

avt.wait(*[Sekunden]***)**
Wartet eine angegebene Zeit an Sekunden (kann ein Dezeimalbruch sein).

Wenn kein Wert angegeben ist, wartet er eine gewisse Zeit.

avt.ticks()
Gibt einen Wert zurück, der jede Millisekunde erhöht wird. Das kann für Zeitsteuerung verwendet werden.

avt.show_avatar()
Zeigt nur den Avatar ohne Sprechblase.

avt.bell()
Gibt einen Ton aus, oder die Anzeige blinkt, wenn die Audio-Ausgabe nicht initialisiert ist.

avt.flash()
Die Anzeige blinkt einmal.

avt.show_image(*Daten***)**
Zeigt ein Bild von Bilddaten. Die *Daten* können ein String mit Bilddaten sein, oder eine Tabelle mit Strings von XPM-Daten. Bei Erfolg wird *true* zurückgegeben, oder *false* im Fehlerfall. Bei Erfolg sollte entweder **avt.wait()** oder **avt.wait_button()** oder **avt.get_key()** aufgerufen werden.

avt.show_image_file(*Dateiname***)**
Lädt ein Bild und zeigt es an. Man kann mit *avt.search()* nach der Datei suchen lassen. Bei Erfolg wird *true* zurückgegeben, oder *false* im Fehlerfall. Bei Erfolg sollte entweder **avt.wait()** oder **avt.wait_button()** oder **avt.get_key()** aufgerufen werden.

avt.subprogram(*function, [arg1, ...]***)**
Ruft die Funktion als Unterprogramm auf.

Bei einer Beendigungsanfrage (dh. wenn man die <Esc>-Taste drückt, oder den Schließ-Knopf des Fensters) kehrt es nur zum Hauptprogramm zurück.

Bei Erfolg werden die Ergebnisse der Funktion zurückgegeben. Bei einer Beendigungsanfrage wird nichts zurückgegeben.

Um ein Unterprogramm aus einer separaten Datei zu starten, kann man **dofile** verwenden.

```
avt.subprogram(dofile, "unterprogramm.lua")
```

avt.optional(*modname***)**
Lädt ein Modul wie **require**, aber das Modul wird nicht benötigt, sondern es ist optional. Das heißt, es ist kein Fehler, wenn das Modul nicht geladen werden kann.

Lua-AKFAvatar braucht nicht initialisiert sein, um diese Funktion zu verwenden.

Unicode

avt.say_unicode(...**)**

Schreibt Text in der Sprechblase mit spezieller Unterstützung für Unicode-Zeichen.

Dieser Befehl ist ähnlich wie **avt.say()**. Strings werden auf die selbe Weise behandelt, abhängig von der eingestellten Kodierung, aber Zahlen werden als Unicode-Codepoint interpretiert, unabhängig von der eingestellten Kodierung.

Man kann das verwenden, um Zeichen zu verwenden, die in der eingestellten Kodierung nicht zur Verfügung stehen. Manchmal ist es aber auch einfach nur einfacher.

Beispiel:

```
avt.say_unicode("Drei Äpfel kosten ",
                tostring(3 * Apfelpreis), 0x20AC,
                ".\n").
```

Wie man hier sieht, muss man **tostring()** verwenden, wenn nummerische Angaben als solche angezeigt werden sollen. Die Zahl *0x20AC* hingegen repräsentiert hier das Euro-Währungszeichen. (Das '0x' leitet hexadezimale Zahlen ein.)

Übrigens werden eine Gruppe von Unicode-Zahlen effektiver verarbeitet als Strings.

avt.printable(*Codepoint***)**

Überprüft ob der angegebene *Codepoint* ein druckbares Unicode-Zeichen darstellt. Es gibt *true* zurück, wenn das der Fall ist, oder *false*, wenn die Schriftart kein druckbares Zeichen dafür hat (es kann trotzdem ein unterstütztes Steuerzeichen sein). Diese Funktion ist unabhängig von der eingestellten Kodierung. Sie kann zusammen mit **avt.say_unicode(...)** eingesetzt werden.

avt.combining(*Codepoint***)**

Überprüft ob der angegebene *Codepoint* ein kombinierendes Unicode-Zeichen darstellt und als solches behandelt wird. Es gibt *true* zurück, wenn das der Fall ist, oder *false*, wenn es nicht als solches unterstützt wird. Das Ergebnis gilt nur für unterstützte Zeichen.

avt.detect_utf8(*String [, Maximallänge]***)**

Prüft, of *String* gültiges UTF-8 (oder ASCII) ist.

Geprüft werden bis zu *Maximallänge* Bytes, aber eine unvollständige Sequenz wird noch abgeschlossen.

avt.toutf8(*Codepoint [, ...]***)**

Nimmt einen oder mehrere angegebene Unicode *Codepoints* und gibt diese als UTF-8 kodierten String zurück.

avt.utf8codepoints(*str***)**

Kann verwendet werden, um alle Unicode Codepoints von einem UTF-8 kodierten String als Zahlen zu erhalten.

Beispiel:

```
for c in avt.utf8codepoints(s) do
  avt.say_unicode(c)
  avt.say(string.format(" = U+%04X", c))
  avt.newline()
end
```

Größen und Positionen

avt.set_balloon_size(*[Höhe] [, Breite]***)**

Legt die Größe der Sprechblase fest. Wenn keine Werte angegeben sind, oder der Wert 0 verwendet wird, wird die maximale Größe verwendet.

avt.set_balloon_width(*[Breite]***)**

Legt die Breite der Sprechblase fest. Ohne Wert, oder bei dem Wert 0 wird das Maximum verwendet.

avt.set_balloon_height(*[Höhe]***)**
Legt die Höhe der Sprechblase fest. Ohne Wert, oder bei dem Wert 0 wird das Maximum verwendet.

avt.get_max_x()
Ermittelt die maximale x-Postion des Cursors in der Sprechblase (also die Breite).

avt.get_max_y()
Ermittelt die maximale y-Postion des Cursors in der Sprechblase (also die Höhe).

avt.where_x()
Ermittelt die x-Position des Cursors in der Sprechblase.

avt.where_y()
Ermittelt die y-Position des Cursors in der Sprechblase.

avt.home_position()
Gibt *true* zurück, wenn sich der Cursor auf der Startposition befindet, oder *false*, wenn nicht. (Das funktioniert auch für rechts-nach-links-Schreibung.)

avt.move_x(*x***)**
Setzt den Curser auf die angegebene *x*-Position.

avt.move_y(*y***)**
Setzt den Curser auf die angegebene *y*-Position.

avt.move_xy(*x, y***)**
Setzt den Curser auf die angegebene *x* und *y* Position.

avt.save_position()
Speichert die aktuelle Cursor-Position.

avt.restore_position()
Stellt eine zuvor gespeicherte Cursor-Postion wieder her.

avt.next_tab()
Setzt den Cursor auf die nächste Tabulator-Position.

avt.last_tab()
Setzt den Cursor auf die vorhergehende Tabulator-Position.

avt.reset_tab_stops()
Setzt die Tabulator-Positionen zurück auf jede achte Spalte.

avt.clear_tab_stops()
Löscht alle Tabulator-Postionen

avt.set_tab(*x, true|false***)**
Setzt oder löscht einen Tabulator an der angegebenen Position *x*.

avt.delete_lines(*Zeile, Anzahl***)**
Löscht die angegebene *Anzahl* an Zeilen, angefangen bei *Zeile*; der Rest wird hochgescrollt.

avt.insert_lines(*Zeile, Anzahl***)**
Fügt die angegebene *Anzahl* an Zeilen ein, angefangen bei *Zeile*; der Rest wird runtergescrollt.

avt.insert_spaces(*Anzahl***)**
Fügt *Anzahl* an Leerzeichen bei der jetzigen Postion ein. Der Rest der Zeile wird weiter gerückt.

avt.delete_characters(*Anzahl***)**
Löscht *Anzahl* an Zeichen an der jetzigen Cursor-Position. Der Rest der Zeile wird zurück gerückt.

avt.erase_characters(*Anzahl***)**
Löscht *Anzahl* an Zeichen. Die Zeichen werden mit Leerzeichen überschrieben.

avt.backspace()
Geht ein Zeichen zurück. Wenn der Cursor am Anfang der Zeile ist, passiert nichts.

Text-Stil

avt.markup(*true*| *false***)**
Setzt den Auszeichnungs-Modus. Im Auszeichnungs-Modus schaltet das Zeichen "_" das Unterstreichen ein oder aus und das Zeichen "*" schaltet den Fettdruck-Modus ein oder aus. Die beiden Zeichen werden im Auszeichnungs-Modus niemals angezeigt!

Man kann immer die Overstrike-Technik verwenden, die keine Zeichen reserviert. Aber die ist schwerer zu benutzen.

avt.underlined(*true*| *false***)**
Schaltet das Unterstreichen ein oder aus.

avt.get_underlined()
Gibt *true* zurück, wenn Unterstreichen eingeschaltet ist, oder *false*, wenn nicht.

avt.bold(*true*| *false***)**
Schaltet den Fettdruck an oder aus.

avt.get_bold()
Gibt *true* zurück, wenn Fettdruck eingeschaltet ist, oder *false*, wenn nicht.

avt.inverse(*true*| *false***)**
Schaltet die invertierte Darstellung an oder aus.

avt.get_inverse()
Gibt *true* zurück, wenn die invertierte Darstellung eingeschaltet ist, oder *false*, wenn nicht.

avt.normal_text()
Setzt normale Einstellungen für Text zurück.

Farben

avt.set_background_color(*Farbe***)**
Setzt die Hintergrundfarbe für den Bildschirm.

Farben können entweder über ihren englischen Namen angegeben werden, oder als RGB-Angabe 6 hexadezimalen Ziffern.

Beispiele:
```
avt.set_background_color("sky blue")
avt.set_background_color(0x8B4513)
avt.set_background_color("#8B4513") --> nicht empfohlen
avt.set_background_color("#555") --> nicht empfohlen
```

avt.set_balloon_color(*Farbe***)**
Setzt die Farbe der Sprechblase.

avt.set_text_color(*Farbe***)**
Setzt die Farbe des Textes.

avt.set_text_background_color(*Farbe***)**
Setzt die Hintergrundfarbe des Textes.

avt.set_text_background_balloncolor()
Setzt die Hintergrundfarbe des Textes auf die Farbe der Sprechblase.

avt.set_bitmap_color(*Farbe***)**
Setzt die Vordergrundfarbe für Bitmaps (einfarbige Grafiken). Der Hintergrund ist immer transparent.

avt.get_color(*Farbnummer***)**
Hole die Farbdefinition für die angegebene Farbnummer.

AKFAvatar hat eine interne Palette mit englischsprachigen Farbnamen, die man verwenden kann. Mit dieser Funktion kann man diese Liste durchgehen. Sie gibt den Namen und die RGB-Definition als String zurück, oder es gibt nichts zurück, wenn die Farbnummer nicht existiert.

avt.colors()

Iterator für interne Farbnamen.

AKFAvatar hat eine interne Palette mit englischsprachigen Farbnamen, die man verwenden kann. Mit dieser Funktion kann man diese Liste mit einer allgemeinen **for**-Schleife durchgehen.

```
require "lua-akfavatar"
for nr, name, rgb in avt.colors() do
  avt.normal_text()
  avt.newline()
  avt.say(string.format("%3d) %5s, %-25s", nr, rgb, name))
  avt.set_text_background_color(name) -- name oder rgb
  avt.clear_eol()
  avt.wait(0.7)
end
avt.wait_button()
```

Wenn man den *rgb*-Wert nicht benötigt, kann man die Variable weglassen.

Interaktion

avt.wait_button()

Wartet bis ein Knopf gedrückt wird.

avt.decide()

Fragt den Benutzer nach einer positiven oder negativen Antwort. Gibt etweder *true* oder *false* zurück.

avt.ask(*[Frage]***)**

Zeigt die *Frage*, falls angegeben, und wartet bis der Benutzer etwas eingibt. Gibt das Eingegebene als String zurück.

Das folgende Beispiel zeigt, wie man die Eingabe einer Zahl erzwingt:

```
require "lua-akfavatar"
avt.save_position()
repeat
  avt.restore_position()
  Zahl = tonumber(avt.ask("Gib eine Zahl ein: "))
until Zahl
avt.say("Die Zahl ist ", Zahl)
avt.wait_button()
```

avt.file_selection(*[Filter]***)**

Startet einen Dateiauswahl-Dialog in der Sprechblase. Am Anfang zeigt er das aktuelle Arbeitsverzeichnis an. Wenn ein Verzeichnis ausgewählt wird, wird das zum Arbeitsverzeichnis. Wenn eine Datei ausgewählt wird, wird der Dateiname zurückgegeben (die sich dann im dann aktuellen Arbeitsverzeichnis befindet), oder im Fehlerfall wird *nil* zurückgegeben.

Der *Filter*, falls angegeben, sollte eine Funktion sein. Sie bekommt einen Dateinamen als Parameter. Die Datei befindet sich immer im aktuellen Arbeitsverzeichnis. Wenn die Filter-Funktion *false* oder *nil* oder nichts zurückgibt, dann wird die Datei nicht angezeigt, ansonsten wird sie angezeigt.

Beispiel:

```
 Textdatei = avt.file_selection(
    function(n)
      return string.find(n,"%.te?xt$")
    end)
```

Natürlich kann für *Filter* auch einfach der Name einer zuvor definierten Funktion angegeben werden.

avt.color_selection()
Startet einen Farbauswahl-Dialog in der Sprechblase. Es werden zwei Strings zurückgegeben: erstens der englische Name für die Farbe und zweitens die hexadezimale RGB-Definition. Beide Werte können für die Farbauswahl verwendet werden.

avt.get_key()
Wartet auf einen Tastendruck und gibt den Unicode-Codepoint des Zeichens zurück. Für einige Funktionstasten werden Werte aus einem Unicode-Bereich für den privaten Gebrauch zurückgegeben.

avt.key_pressed()
Prüft, ob eine Taste gedrückt wurde. Um den Tasten-Code abzuholen kann man **avt.get_key()** verwenden.

avt.clear_keys()
Löscht den Tastatur-Buffer

avt.push_key(*Codepoint***)**
Simuliert einen Tastendruck

avt.navigate(*buttons***)**
Zeigt eine Navigationsleiste mit den angegebenen Knöpfen.

Für die Knöpfe kann man in einem String die folgenden Zeichen verwenden:

l:	links
r:	rechts (abspielen)
d:	runter
u:	hoch
x:	abbrechen
f:	(schnell)vorwärts
b:	(schnell)rückwärts
p:	Pause
s:	Stop
e:	Auswurf
*:	Kreis (Aufnahme)
+:	Plus (hinzufügen)
-:	Minus (entfernen)
?:	Hilfe
’ ’:	Platzhalter (kein Knopf)

Eine Taste mit einem dieser Zeichen zu drücken wählt diesen aus. Für die Richtungen kann man auch die Pfeiltasten verwenden. Die <Pause>-Taste gibt "p" zurück. Die <Hilfe>-Taste oder <F1> geben "?" zurück.

Es wird das entsprechende Zeichen zurückgegeben oder eine Zahl.

Wenn eine Audio-Ausgabe endet, während diese Funktion aktiv ist, wird automatisch entweder "f" (vorwärts) oder "s" (Stop) ausgewählt. Falls beides vorhanden ist, hat "f" Vorrang.

avt.menu(*Menüpunkte***)**

avt.long_menu(*Menüpunkte***)**

Zeigt ein Menü mit den angegebenen *Menüpunkte*n. Die *Menüpunkte* können zum Einen ein Array mit Strings sein. Dann wird die Nummer (Position) des ausgewählten Menüpunktes zurückgegeben.

Oder *Menüpunkte* können ein Array mit weiteren Arrays sein. Die inneren Arrays müssen dann mit einem String anfangen, gefolgt von einem oder mehreren Egebnissen. Die Ergebnisse können jeder beliebige Lua-Typ sein, einschließlich Funktionen.

Das Menü fängt in der Zeile der aktuellen Cursor-Position an. Dadurch kann man eine Überschrift vor das Menü setzen.

```
avt.clear()
avt.say("Bitte das Liebligsessen auswählen:\n")
local Menuepunkt = avt.long_menu {
  "Chicken",
  "Chips",
  "Pizza",
  "Spinach"}
```

avt.choice(*Anfanszeile, Einträge [, Taste] [, zurück] [, vorwärts]***)**

Diese Funktion kann für Menüs verwendet werden. Es ist eine grundlegendere Funktion, als **avt.menu()**. Es gibt die Nummer den ausgewählten Menüpunktes zurück.

Anfangszeile:
: die Zeile, in der die Auswahl beginnt

Einträge:
: Anzahl der Einträge (Zeilen)

Taste: Anfangstaste, wie "1" oder "a", 0 für keine Tastenunterstützung

zurück: auf *true* zu setzen, wenn der erste Eintrag eine zurück-Funktion hat

vorwärts:
: auf *true* zu setzen, wenn der letzte Eintrag eine weiter-Funktion hat

Audio-Ausgabe

avt.start_audio()

Startet das Audio-Untersystem.

Bei Erfolg gibt es *true* zurück, im Fehlerfall gibt es *nil* und eine Fehlermeldung zurück.

avt.load_audio_file(*[Dateiname [,Abspielmodus]]***)**
avt.load_base_audio_file(*[Dateiname [,Abspielmodus]]***)**

Liest Audio-Daten von einer Datei ein. Man kann mit **avt.search()** nach der Datei suchen lassen.

Lua-Module können Unterstützung für weitere Audio-Formate zu **avt.load_audio_file()** hinzufügen. (Das tut zum Beispiel das Modul **akfavatar-vorbis**).

Wenn kein *Dateiname* angegeben ist, oder der *Dateiname* ist *nil* oder ein leerer String, wird ein Audio-Element mit Stille zurückgegeben, das heißt, man kann die Methoden aufrufen, aber es wird kein Klang ausgegeben.

Abspielmodus kann entweder "load", "play" oder "loop" sein.

Bei Fehlern wird *nil* und eine Fehlermelung zurückgegeben. (Anmerkung: in Version 0.19.0 wurde dann auch eine Stille zurückgegeben.)

avt.load_audio_stream(*Datei [,Größe [,Abspielmodus]]***)**
avt.load_base_audio_stream(*Datei [,Größe [,Abspielmodus]]***)**

Liest Audio-Daten von einer geöffneten Datei. Die Datei muss durchsuchbar sein.

Lua-Module können Unterstützung für weitere Audio-Formate zu **avt.load_audio_stream()** hinzufügen. (Das tut zum Beispiel das Modul **akfavatar-vorbis**).

Wenn keine Größe angegeben ist, wird davon ausgegangen, dass die Audio-Daten bis zum Ende der Datei gehen.

Abspielmodus kann entweder "load", "play" oder "loop" sein.

Bei Fehlern wird *nil* und eine Fehlermelung zurückgegeben.

avt.load_audio(*[Audiodaten [,Abspielmodus]]***)**
avt.load_base_audio(*[Audiodaten [,Abspielmodus]]***)**
Liest Audiodaten aus einem String. Ansonsten das selbe, wie **avt.load_audio_file()**.

Lua-Module können Unterstützung für weitere Audio-Formate zu **avt.load_audio()** hinzufügen. (Das tut zum Beispiel das Modul **akfavatar-vorbis**).

Wenn keine *Audiodaten* angegeben sind, oder *Audiodaten* ist *nil* oder ein leerer String, wird ein Audio-Element mit Stille zurückgegeben, das heißt, man kann die Methoden aufrufen, aber es wird kein Klang ausgegeben.

Abspielmodus kann entweder "load", "play" oder "loop" sein.

Bei Fehlern wird *nil* und eine Fehlermelung zurückgegeben. (Anmerkung: in Version 0.19.0 wurde dann auch eine Stille zurückgegeben.)

avt.silent()
Gibt ein stilles Audio-Element zurück, das heißt, man kann die Methoden aufrufen, aber es wird kein Klang ausgegeben.

Beispiel:
```
audio = avt.load_audio_file(Dateiname) or avt.silent()
```

In diesem Beispiel bekommt man einen stillen Klang, wenn die Datei nicht gelesen werden konnte.

avt.alert()
Gibt ein Pseudo-Audio-Element zurück, welches **avt.bell()** aufruft, wenn man es abspielt.

avt.audio_playing(*[Audiodaten]***)**
Überprüft, ob Audiodaten gerade abgespielt werden. Wenn *Audiodaten* angegeben ist und nicht *nil* ist, dann wird überprüft, ob die angegebenen Audiodaten abgespielt werden. Das kann man übrigens auch mit *audio***:playing()** überprüfen.

avt.wait_audio_end()
Wartet, bis die Audio-Ausgabe beendet ist.

Dadurch wird auch eine Audio-Schleife beendet, aber es spielt halt noch bis zum Ende des aktuellen Klanges.

avt.stop_audio()
Stoppt die Audio-Ausgabe sofort.

avt.pause_audio(*true*|*false***)**
Die Audio-Ausgabe wird pausiert (*true*) oder weiter gespielt (*false*)

*audio***:play()**
*audio***()** Spielt die Audiodaten *audio* ab. Die Audiodaten *audio* müssen mit **avt.load_audio_file()** oder **avt.load_audio_string()** geladen worden sein.

Es kann nur ein Klang gleichzeitig abgespielt werden. Wenn man einen anderen Klang abspielt, wird der vorherige dadurch abgebrochen. Man kann **avt.wait_audio_end()** verwenden, um Klänge nacheinander abzuspielen.

Man kann den Klang auch abspielen, indem man die Audio-Variable wie eine Funktion aufruft.

```
abspielen = function (Dateiname)
  local Klang = avt.load_audio_file(avt.search(Dateiname))
  if Klang then Klang:play() end
```

```
end
```

audio:**loop()**
: Spielt die Audiodaten *audio* in einer Schleife ab. Die Audiodaten *audio* müssen mit **avt.load_audio_file()** oder **avt.load_audio_string()** geladen worden sein.

  Das ist zum Beispiel für eine kurze Musik-Sequenz nützlich.

  Man kann die Audio-Schleife mit **avt.wait_audio_end()** oder **avt.stop_audio()** beenden.

audio:**playing()**
: Überprüft, ob diese *audio*-Daten gerade abgespielt werden. Die Audiodaten *audio* müssen mit **avt.load_audio_file()** oder **avt.load_audio_string()** geladen worden sein.

  Diese Funktion ist identisch mit **avt.audio_playing(audio)**.

audio:**free()**
: Gibt die *audio*-Daten frei. Falls diese *audio*-Daten gerade abgespielt werden, wird die Audio-Ausgabe abgebrochen.

  Audiodaten werden auch vom Garbage Collector freigegeben.

avt.set_audio_end_key (*Taste***)**
: Definiere eine Taste, die automatisch gedrückt werden soll, wenn die Audio-Ausgabe endet. Die *Taste* sollte als Zahl für den Unicode-Codepoint angegeben werden. Durch den Wert *0* kann man das wieder abschalten.

  Die Funktion gibt den voher gesetzten Wert zurück.

avt.quit_audio()
: Das Audio-Untersystem beenden.

  Das ist bei normalen Programmen nicht nötig. Diese Funktion sollte nur verwendet werden, wenn man weiß, was man tut.

Dateisystem

avt.dirsep
: Diese Variable enthält das Verzeichnis-Trennzeichen des Systems; entweder "/" oder "\".

avt.datapath
: Diese Variable enthält den Standardsuchpfad für die Funktion **avt.search()** (siehe unten). Verzeichnisse werden durch Semikola getrennt. Es gibt keine Muster, wie in den Pfaden für Lua-Module, sondern es werden nur Verzeichnisse angegeben. Diese Variable wird entweder durch die Umgebungsvariable *AVTDATAPATH* initialisiert, oder sie bekommt eine systemspezifische Vorgabeeinstellung.

avt.search(*Dateiname [,Pfad]***)**
: Sucht eine Datei mit dem angegebenen *Dateiname*n im angegebenen *Pfad*. Wenn kein *Pfad* angegeben ist, wird die Variable *avt.datapath* verwendet.

  Wenn die Datei gefunden wurde, wird der vollständige Pfad der Datei zurückgegeben. Wenn die Datei nicht gefunden wurde, wird *nil* und eine Fehlermeldung zurückgegeben.

avt.get_directory()
avt.getcwd()
: Gibt das aktuelle Arbeitsverzeichnis zurück. Im Fehlerfall wird *nil* und eine Fehlermeldung zurückgegeben.

avt.set_directory(*directory***)**
avt.chdir(*Verzeichnis***)**
: Setzt das Arbeitsverzeichnis auf *Verzeichnis*. Wenn *Verzeichnis nil* ist, ein Leerstring oder nicht angegeben ist, wird nichts ausgeführt.

  Gibt bei Erfolg *true* zurück oder bei einem Fehler *nil* und eine Fehlermeldung.

Beispiel:

```
avt.set_directory(os.getenv("HOME") or os.getenv("USERPROFILE"))
```

avt.directory_entries(*[Verzeichnis]***)**

Hole eine Liste von Verzeichniseinträgen vom angegebenen *Verzeichnis* oder dem aktuellen Arbeitsverzeichnis, wenn keins angegeben ist.

Bei Erfolg gibt es ein Array mit den Verzeichniseinträgen zurück und die Anzahl der Einträge. Im Fehlerfall wird *nil* und eine Fehlermeldung zurückgegeben.

Die Liste enthält sowohl normale Dateinamen, einschließlich verstecken Dateien, Unterverzeichnisse und andere Arten von Einträgen. Die Einträge "." oder ".." sind nicht mit drin.

Achtung: Die Namen sind in einer systemspezifischen Kodierung. Um die Namen anzeigen zu können, muss man entweder die Kodierung der Anzeige mit **avt.encoding("SYSTEM")** ändern, oder die Namen wie folgt konvertieren: **avt.say(avt.recode(name, "SYSTEM"))**.

avt.entry_type(*Eintrag***)**

Ermittle die Art eines Verzeichniseintrags und seine Größe.

Bei Erfolg gibt es die Art des Verzeichniseintrags als String zurück und die Größe als Zahl. Die Art ist entweder "file", "directory", "character device", "block device", "fifo", "socket" oder "unknown".

Im Fehlerfall wird *nil* und eine Fehlermeldung zurückgegeben.

Symbolische Links werden ausgewertet. Das bedeutet, man bekommt die Art des resultierenden Eintrags. Kaputte Links werden wie nicht existierende Einträge behandelt.

Verschiedenes

avt.language

Diese Variable enthält einen Sprachcode für Meldungen. Es sollte sich um eine Kennung mit zwei Buchstaben gemäß ISO 639-1 handeln. Wenn die Sprache nicht ermittelt werden konnte, ist die Variable nicht gesetzt.

avt.translate(*Text***)**

Übersetzt den *Text*, falls möglich.

Wie man Übersetzungen angeben kann, steht im Abschnitt **Übersetzungen**.

avt.recode(*String, Quellkodierung [,Zielkodierung]***)**

Konvertiert den angegebenen *String*, der in *Quellkodierung* kodiert ist, in einen String der in *Zielkodierung* kodiert ist. Wenn nur eine Kodierung angegeben ist, kovertiert es in die aktuell gesetzte Kodierung. Falls man von der aktuellen Kodierung in etwas anderes kovertieren will, kann man *nil* für die *Quellkodierung* angeben.

Um in die oder aus der Standardkodierung des Systems umzukodieren (zum Beispiel für Dateinamen), kann man einen leeren String (*""*) oder *"SYSTEM"* angeben.

Gibt den kodierten String zurück, oder *nil*, bei schweren Fehlern.

Achtung: Nach UTF-8 zu konvertieren ist immer möglich. Nach anderen Kodierungen zu konvertieren, kann jedoch fehlschlagen. Zeichen, die nicht konvertiert werden können, werden durch das Steuerzeichen *SUB* (*"\x1A"*) ersetzt. Um sicher zu stellen, dass die Konvertierung verlustfrei ablief, kann man überprüfen, dass dieses Zeichen im Ausgabestring nicht vorkommt.

avt.right_to_left(*true|false***)**

Aktiviert, oder deaktiviert den rechts-nach-links-Schreibmodus.

Achtung: Dies ist noch experimentell und funktioniert nur eingeschränkt.

avt.set_flip_page_delay(*[delay]***)**

Setzt eine Verzögerung für **avt.flip_page()** oder "\f". Ohne Angabe eines Wertes wird auf die Vorgabe zurückgesetzt. Der Wert 0 schaltet die Verzögerung ganz aus.

avt.activate_cursor(*true*| *false***)**
Legt fest, ob der Cursor angezeigt wird, oder nicht.

avt.clear_screen()
Löscht den gesamten Bildschirm, bzw. das Fenster (nicht nur die Sprechblase!).

avt.clear_down()
Löscht von der Cursor-Postition nach unten im Anzeigebereich. Wenn noch keine Sprechblase angezeigt wird, wird sie gezeichnet.

avt.clear_eol()
Löscht das Ende der Zeile (abhängig von der Text-Richtung).

avt.clear_bol()
Löscht den Anfang der Zeile (abhängig von der Text-Richtung).

avt.clear_line()
Löscht die Zeile.

avt.clear_up()
Löscht von der Cursor-Postition nach oben im Anzeigebereich. Wenn noch keine Sprechblase angezeigt wird, wird sie gezeichnet.

avt.reserve_single_keys(*true*| *false***)**
Reserviert Einzeltasten, wie <ESC> oder <F11>.

avt.switch_mode(*mode***)**
Ändert den Fenster-Modus. Man kann es entwerder auf *"window"*, oder *"fullscreen"* setzen.

(Die Modi *"auto"* und *"fullscreen no switch"* funktionieren hiermit nicht.)

avt.get_mode()
Gibt den Fenster-Modus zurück (siehe **avt.switch_mode (mode)**).

avt.toggle_fullscreen()
Schaltet den Vollbild-Modus ein oder aus.

avt.update()
Aktualisiert alles und reagiert auf Ereignisse. Dies sollte innehalb von Schleifen ausgeführt werden, während das Programm mit anderem beschäftigt ist.

avt.credits(*text, centered***)**
Zeigt einen Abspann.

Wenn der zweite Parameter *true* ist, wird jede Zeile zentriert.

avt.viewport(*x, y, width, height***)**
Setzt einen Anzeigebereich (einen Unterbereich des Textbereiches). Die obere linke Ecke hat die Koordinaten 1, 1.

avt.set_scroll_mode(*Modus***)**
Setzt den Rollmodus, dh. wie er reagiert, wenn man hinter der letzten Zeile weiter schreibt. Der *Modus* ist entweder -1 für "nichts tun" oder 0 für "Seite umblättern" oder 1 für "hochrollen".

avt.get_scroll_mode()
Gibt den Rollmodus zurück (siehe **avt.set_scroll_mode()**).

avt.newline_mode(*true*| *false***)**
Wenn der Neue-Zeile-Modus aktiviert ist (Vorgabe), dann setzt ein Zeilenvorschub-Zeichen den Cursor an den Anfang einer neuen Zeile. Wenn er aus ist, geht der Cursor in die nächste Zeile, bleibt aber in der selben horizontalen Position.

avt.set_auto_margin(*true*| *false***)**
Setzt den Modus für automatischen Zeilenumbruch, dh. ob eine neue Zeile angefangen werden soll, wenn der Text nicht in eine Zeile passt.

avt.get_auto_margin()
Gibt den Modus für automatischen Zeilenumbruch zurück.

avt.set_origin_mode(*true*| *false***)**
Setzt den Ursprungs-Modus. Wenn der Ursprungs-Modus eingeschaltet ist, sind die Koordinaten 1, 1 immer oben links in der Sprechblase, auch dann, wenn der Anzeigebereich (viewport) nicht dort beginnt. Wenn der Ursprungs-Modus ausgeschaltet ist, sind die Koordinaten 1, 1 oben links im Anzeigebereich (viewport).

avt.get_origin_mode()
Gibt den Ursprings-Modus zurück (siehe **avt.set_origin_mode**).

avt.set_mouse_visible(*true*| *false***)**
Legt fest, ob der Mauszeiger sichtbar sein soll, oder nicht.

Anmerkung: Wenn die Anwendung in einem Fenster läuft, gilt das nur, wenn sich der Mauszeiger innerhalb des Fensters befindet.

avt.lock_updates(*true*| *false***)**
Blockiert Aktualisierungen innerhalb der Sprechblase. Das kann man verwenden um die Geschwindigkeit zu erhöhen.

avt.version()
Gibt die Version von AKFAvatar als String zurück.

avt.copyright()
Gibt die Copyright-Meldung für AKFAvatar als String zurück.

avt.license()
Gibt die Lizenz-Meldung für AKFAvatar als String zurück.

avt.quit()
Beendet das AKFAvatar Untersystem (schließt das Fenster). Das Audio-Untersystem wird ebenfalls beendet.

Diese Funktion wird nicht in normalen Programmen benötigt. Man sollte sie nur verwenden, wenn das Programm ohne sichbares Fenster weiter laufen soll.

avt.launch(*Programm [,Argumente ...]***)**
Beendet AKFAvatar und führt das angegebene *Programm* aus. Diese Funktion kehrt niemals zurück. Entweder das Programm läuft, oder es wird ein fataler Fehler angezeigt.

Wenn man Fehler mit **pcall** abfangen will, muss man danach AKFAvatar neu initialisieren...

Übersetzungen

Um Übersetzungen für Lua-AKFAvatar Skripte zu schreiben, muss man erstmal die Variable **avt.translations** definieren. Es handelt sich dabei um eine verschachtelte Tabelle. Nun, die ist schwer zu beschreiben, aber das Beispiel-Skript weiter unten sollte es verständlich machen.

Die zu verwendende Sprache wird durch die Variable **avt.language** festgelegt. Diese Variable sollte von Lua-AKFAvatar automatisch initialisiert worden sein, sie kann aber auch im Skript geändert werden. Sie enthält eine Sprach-Kennung mit zwei Buchstaben gemäß ISO 639-1.

Die Funktion **avt.translate(***Text***)** gibt dann den übersetzten Text zurück. Falls keine Übersetzung zur Verfügung steht, wird der Text unverändert zurück gegeben.

Es ist ratsam, einen lokalen Alias namens **L** für **avt.translate** anzulegen:

```
  local L = avt.translate
```

Dann kann man einfach eine String-Konstante damit einleiten.

Beispiel:

```
local avt = require "lua-akfavatar"

avt.encoding("UTF-8")
```

```
avt.translations = {

  ["Hello world!"] = {
    es="¡Hola mundo!",
    fr="Bonjour le monde!",
    de="Hallo Welt!",
    sv="Hej Världen!",
    },

  ["That’s live!"] = {
    de="So ist das Leben!",
    fr="C’est la vie!" },
}

local L = avt.translate

-- avt.language = "de"

avt.start()
avt.avatar_image("default")
avt.tell(L"Hello world!", "\n", L"That’s live!");
avt.wait_button ()
```

Tipps:

– Obwohl es nicht notwendig ist, sollte man Englisch als Ausgangssprache verwenden.

– Wenn man Text mit Variablen dazwischen hat, ist es kein guter Ansatz, den Text in Teile aufzuteilen. Es ist besser einen Format-String für **string.format()** zu definieren.

– Der übersetzbare String muss exakt übereinstimmen. Bitte daran denken, wenn man die Strings im Programm abändert, dass man dann auch die Übersetzungstabelle anpassen muss!

– Der String kann auch ein Dateiname für eine Textdatei oder eine Sprachaufnahme sein.

Übrigens, diese Implementierung wurde von GNU gettext inspiriert.

SIEHE AUCH

lua-akfavatar(1) **lua**(1) **akfavatar-graphic**(3) **akfavatar-term**(3) **akfavatar.utf8**(3)
http://lua.coders-online.net/
http://www.lua.org/manual/5.2/
http://akfavatar.nongnu.org/manual/