

# **GatewayLibrary**

Michaela Prüß

Copyright © 1998, 1999 by Michaela Prüß, All Rights Reserved

COLLABORATORS

	TITLE : GatewayLibrary		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY	Michaela Prüß	May 24, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>GatewayLibrary</b>	<b>1</b>
1.1	Gateway.Library V 17.1.1 - Funktionen	1
1.2	Wichtig!!!	1
1.3	Copyright	2
1.4	GATEWAY	3
1.5	GateRequest	3
1.6	ltofa	3
1.7	trim	4
1.8	rtrim	4
1.9	lset	5
1.10	lsetmin	5
1.11	string	5
1.12	instr	6
1.13	upstr	6
1.14	lowstr	7
1.15	set	7
1.16	midstr	7
1.17	date_to_day	8
1.18	date_to_zahl	8
1.19	time_to_zahl	8
1.20	kill_ansi	9
1.21	newer	9
1.22	swapmem	10
1.23	memncmp	10
1.24	StrCaseCmp	10
1.25	trim_includes	11
1.26	mail_trim	11
1.27	newstr	11
1.28	wordwrp	12
1.29	fnSplitt	13

1.30	fn_build . . . . .	13
1.31	addval . . . . .	14
1.32	strdup . . . . .	14
1.33	index . . . . .	14

## Chapter 1

# GatewayLibrary

### 1.1 Gateway.Library V 17.1.1 - Funktionen

**Ganz wichtig** Vor Benutzung lesen! **Copyright** Copyright, Nutzung! **Name Gateway** Gateway-5 ist nicht Gateway 2000!

Funktionen:

**GateRequest** Einfacher Requester

**trim** Leerzeichen entfernen **rtrim** Rechtsbündige Leerzeichen entfernen **trim\_includes** trim für Include-File-Namen **mail\_trim** trim für eMail

**set** String auf feste Länge **lset** String auf feste Länge setzen **lsetmin** String auf eine Mindestlänge setzen

**instr** Teilstring suchen **midstr** Teilstring ausschneiden

**newstr** Teilstring im Hauptstring ersetzen **wordwrp** Wordumbruch Zeilenende **kill\_ansi** Ansicodes entfernen

**fnSplitt** Vollen Datei u. Pfadnamen aufsplitten **fn\_build** Neuen Dateinamen bilden

**time\_to\_zahl** Zeit in Zahl wandeln **date\_to\_zahl** Datum in Zahl wandeln **date\_to\_day** Tag im Jahr (ca.) berechnen **addval** Zahlenstring und Zahl zu neuem String

**ltofa** Formatierte Zahlausgabe

**string** String mit Zeichen füllen **newer** Datum+Zeit neuer als 2.Paar? **upstr** String in Großbuchstaben **lowstr** String in Kleinbuchstaben

**StrCaseCmp** Strings vergleichen **strdup** Dupliziere String

**swapmem** Speicherbereiche vertauschen **memncmp** Speicherbereiche vergleichen

**index** Suchen eines Zeichens im String

### 1.2 Wichtig!!!

"Wichtig! Unbedingt zuerst lesen"

Die Gateway.Library ist eine Sammlung von (meist String-)Funktionen die im Gateway-BBS-System verwendung finden. Die Library entstand erst lange nach dem Kernprogramm das über Jahre gewachsen ist. Die Funktionen die sich jetzt in der Library befinden waren im Programm verstreut und sind teilweise mit erheblichen Zeitabständen entstanden. Daher sind die Variablennamen nicht einheitlich und manche Funktion würde heute sicher- lich anders realisiert. Fakt ist, das Programm nutzt die meisten der Funktionen sehr stark und das an vielen verschiedenen Stellen. Dadurch ist es kaum möglich, eine Funktion in der Schnittstelle zu verändern.

Neben diesen alten Routinen finden sich noch StringsFunktionen, die allgemein bekannt sein dürften, wie z.B. 'strdup'. Diese Funktionen werden von Gateway selbst oder von den Gateway-Tools benutzt. Da sie im vbcc-Compiler nicht vorhanden waren

habe ich sie in der Library untergebracht. Ich bin drauf und dran, die allgemeinen C-Standard-Stringfunktionen komplett in die Library zu verlegen. Man muß sie ja nicht nutzen, könnte es jedoch und ich habe später bei den geplanten Crosscompilaten keine Probleme. Mal schauen.

Gateway verwendet heute eine Link-Library mit sehr vielen, oft recht komplexen Funktionen, die zu sehr großen Teilen Amiga-DOS direkt oder indirekt benötigen. Da einige dieser Funktionen in den PMM-Programmen benutzt werden und diese Funktionen teilweise wiederum die Gateway.Library benötigen, geht hiermit erstmals eine Gateway.Library ohne das BBS-System heraus. Und da ich es ausgesprochen blöd finde, wenn eine Library undokumentiert im System gammelt, während irgendjemand vielleicht irgendwas davon gebrauchen könnte, habe ich hier eine Dokumentation der Library beigelegt. Auch wenn es kaum anzunehmen ist, das sich viele Leute dafür interessieren.

Etliche Funktionen lassen sich auch mit einem `sprintf` oder ähnlichen Befehlen lösen. Das hat nur den Nachteil, das meist eine Buffervariable notwendig ist und es sich bei diesen Funktionen um Funktionen mit `varargs` handelt was alleine schon langsam genug ist. Die Formaterkennung und das nachfolgende Einsetzen der Variable in den String sind garantiert auch nochmals langsamer als die Funktionen der Library. Von unnötigen CPU-Belastungen mal ganz zu schweigen. Während das im einfachen Benutzerbetrieb des Rechners ja noch egal sein mag, im Multi-User-Betrieb einer Mailbox ist das nicht mehr brauchbar. Aus gleichen Gründen haben die Stringfunktionen eine feste interne Puffergröße. Strings mit mehr als 4096 Bytes lassen die Funktionen crashen. Ausgenommen die `'mem'`-Funktionen, `upstr`, `lowstr` und `StrCaseCmp`, die haben keine Limits.

Einige Funktionen, mögen zu dem Zeitpunkt da diese Dokumentation gelesen wird, längst überflüssig geworden sein. Oder es wird nicht mit `vbcc` gearbeitet und die Funktionen sind vorhanden. Fest steht, zum heutigen Zeitpunkt gibt es die Funktionen in `vbcc` nicht, zum Teil gab es sie jedoch auf dem Compiler, auf dem alles mal entwickelt wurde und daher wurden sie genau nachempfunden. Das trifft z.B. auf `swapmem` zu, eine Funktion die zum Teil unter dem Namen `swab` bei anderen Compilern auftaucht. Funktionell ist es auf jeden Fall identisch.

Die Library ist Bestandteil von Gateway-5 und unterliegt den gleichen Copyright's. Jede Benutzung erfolgt ausschließlich auf eigene Gefahr. Für die Richtigkeit der hier gemachten Angaben wird keinerlei Garantie übernommen. Desweiteren besteht keine Anspruch darauf, das die Funktionen in zukünftigen Libraryfunktionen erhalten oder unverändert bleiben.

## 1.3 Copyright

"Copyright und Nutzungsbedingungen!"

Die gateway.library ist Copyright by Michaela Prüß. Alle Rechte vorbehalten.

Solange die Library in unveränderten Zustand verbleibt, darf sie frei kopiert werden. Unveränderter Zustand heißt auch, das die der Library beiliegenden Files unverändert bei der Library verbleiben müssen.

Dies sind zur Zeit (12/98): Diese Dokumentation (GatewayLibrary.Guide) Das .FD-File 'Gateway\_Lib.FD'. Der Amiga-DOS-Batch '\_Gateway\_Fd\_Script' mit dem aus der .fd-Datei die Pragma- und Amicall-Definitionen erzeugt werden. Die Includefiles 'gateway\_lib.h' und 'gateway\_protos.h'

Jede Veränderung an den Files ist untersagt und führt sofort zum Verlust der Nutzungslizenz.

Ausnahme: Da Compiler die im Standard Funktionen enthalten, welche in der Library definiert sind, kann es zu Problemen mit den Library-Includes kommen. In diesem Fall dürfen diese beiden Datei lokal angepasst werden. Wer eine solche Anpassung vornimmt schickt die Files bitte an mich zurück, ich nehme sie ggf. in das Archiv auf.

Eine Gewährleistung, gleich welcher Art, wird weder für die Library noch für den Inhalt der sonstigen Files übernommen. Schadensersatz ist generell ausgeschlossen, selbst wenn durch die Nutzung Datenverluste entstehen sollten. Die benutzung erfolgt generell auf eigenes Risiko.

Änderungen an der Library können jederzeit vorgenommen werden.

Copyright's, Lizenzen und Sources dritter:

Die Library basiert zum Teil auf den Souces der example.library von Andreas Kleinert. Zum Teil mußten einige Dinge darin verändert werden da sich meine `vbcc`-Version augenscheinlich nicht kompatibel zu der Version verhält für die die Library irgendwann angepasst wurde. Falls dies hier von Andreas Kleinert gelesen wird: Ich habe aus Zeitgründen keine Ursachen erforscht, sondern lediglich alles entfernt, was hier zu Komplikationen geführt hat. Wenn mehr Zeit ist werde ich versuchen die "saubere" Library anzupassen und dann die Änderungen melden.

`vbcc` ist Copyright bei Volker Barthelmann (und zum Teil Frank Wille).

Gateway-5, gateway.library und Pmm sind Copyright bei Michaela Prüß.

Amiga ist ein Warenzeichen von AMIGA / GATEWAY 2000.

## 1.4 GATEWAY

"Ein Name, zwei Dinge!"

Dies in (nicht nur) eigener Sache:

Als die Firma GATEWAY 2000 bzw. jetzt nur noch GATEWAY die Rechte am Amiga gekauft hat, war das so ziemlich der größte Zufall der sich nur denken läßt. Zu diesem Zeitpunkt war meine BBS-Software mit dem Namen GATEWAY schon 8 Jahre lang im Vertrieb. Erst seit der 2.ten Version, einem Redesign des ersten Programmes, trägt der Name einen numerischen (römisch-numerischen Zusatz). Bis heute gab/gibt es: GATEWAY GATEWAY-II GATEWAY-III GATEWAY-IV Gateway-5

Es soll hier an dieser Stelle ganz klar gesagt werden: Dies ist eine BBS-Software die in keiner Weise etwas mit dem jetzigen Inhaber des Amiga's zu tun hat. Da ich mich aber auf einen rechtlich gültigen Namensschutz stützen kann werde ich den Namen der BBS-Software nicht abändern. Bis jetzt hat das auch niemand verlangt.

Aber um Problemen und Verwechslungen, gleich welcher Art, im Voraus zu begegnen, hänge ich inzwischen allen meinen Veröffentlichungen diese Information an.

Ich erhebe auch lediglich Titelschutz auf den Namen GATEWAY als BBS-Software und als Fernseh-Sendung (das interaktive TV-Programm GATEWAY lief über ein Jahr lang im Berliner Kabelnetz). Hier sei noch erwähnt, das der Mitträger der TV-Sendung ebenfalls die Rechte hat. Das ist "Der freie Kanal" in Berlin.

## 1.5 GateRequest

```
"ULONG GateRequest(UBYTE *title_d1,UBYTE *body,UBYTE *gadgets);"
```

Zeigt einen einfachen Requester via EasyRequest

Diese Funktion dient zum Test ob die Library anläuft.

Aufrufwerte:

-----

UBYTE \*title\_d1 - Überschrift des Requesters

UBYTE \*body - Text des Requesters

UBYTE \*gadgets - Beschriftung der Auswahlknöpfe

Rückgabe:

-----

ULONG - Nummer des gewählten Gadgets

## 1.6 Itofa

```
"char *Itofa(char *tx_d1,ULONG l);"
```

(Long TO formatted ascii)

Die Funktion formatiert eine Zahl (ähnlich einem 'PRINT USING') mit Punkten an den nach der 3. ,6. und 9. Ziffer.

Die Rückgabe hat das Muster "#.###.###.###"

Nicht benutzte Positionen sind mit Leerzeichen aufgefüllt, die Rückgabe hat immer die Länge 13 + Nullbyte.

Beispiele:

---

1234567890 => '1.234.567.890'

482 => ' 482'

48526 => ' 48.526'

Aufrufwerte:

-----

char \*tx\_d1 - Pointer auf ein mindestens 14 Zeichen großes Char-Feld. In dem Feld wird die Eingabe aufbereitet zurückgeliefert.

ULONG l - Die Zahl, die formatiert werden soll.

Rückgabe:

-----

char \* - Pointer auf tx\_d1

## 1.7 trim

"void trim(UBYTE \*trptr);"

Funktion zum Entfernen von Leerzeichen am Anfang und Ende von Strings. Entfernt werden die Ascii-Codes 32, 160 und 9.

Leerzeichen, geschiftete Leerzeichen und Tabulatoren werden abgeschnitten. Diese Funktion dient zum Säubern von Zeichenketten die aus Dateien eingelesen werden.

Aufrufwerte:

-----

UBYTE \*trptr - Pointer auf den String der bereinigt wird. Nach dem Funktionsaufruf ist der String bereinigt.

Rückgabe:

-----

N/A

## 1.8 rtrim

"void rtrim(UBYTE \*trptr);"

Gleiche Funktion wie trim, es wird nur der rechte Teil des Strings bearbeitet. Führende Leerzeichen/Tab's bleiben. Entfernt werden die Ascii-Codes 32, 160 und 9.

Leerzeichen, geschiftete Leerzeichen und Tabulatoren werden abgeschnitten. Diese Funktion dient zum Säubern von Zeichenketten die aus Dateien eingelesen werden.

Aufrufwerte:

-----

UBYTE \*trptr - Pointer auf den String der bereinigt wird. Nach dem Funktionsaufruf ist der String bereinigt.

Rückgabe:

-----

N/A

---

## 1.9 lset

```
"void lset(UBYTE *lbuff, int slen);"
```

String formatieren auf eine festgelegte Anzahl Stellen.

Ist der String kürzer als gewünscht werden an Anfang Leerzeichen eingefügt.

Ist der String länger als gewünscht wird er rechts abgeschnitten.

Zuvor wird (und deshalb würde sprintf nicht das gleiche Ergebnis liefern) ein 'trim' durchgeführt.

Es wird also der wirkliche String ohne Leerzeichen am Anfang oder Ende formatiert.

Aufrufwerte:

-----

UBYTE \*lbuff - Pointer auf den String der bearbeitet wird. Nach dem Funktionsaufruf ist der String ggf. geändert.

int slen - Länge die der String nach Aufruf der Funktion haben soll.

Rückgabe:

-----

N/A

## 1.10 lsetmin

```
"void lsetmin(UBYTE *lbuff, int slen);"
```

String formatieren auf eine festgelegte Mindestanzahl Stellen.

Ist der String kürzer als gewünscht werden an Anfang Leerzeichen eingefügt.

Zuvor wird (und deshalb würde sprintf nicht das gleiche Ergebnis liefern) ein 'trim' durchgeführt.

Es wird also der wirkliche String ohne Leerzeichen am Anfang oder Ende formatiert.

Im Gegensatz zu lset wird der String aber in keinem Falle gekürzt.

Aufrufwerte:

-----

UBYTE \*lbuff - Pointer auf den String der bearbeitet wird. Nach dem Funktionsaufruf ist der String ggf. geändert.

int slen - Länge die der String nach Aufruf der Funktion mindestens haben soll.

Rückgabe:

-----

N/A

## 1.11 string

```
"void string(UBYTE *spstr, int num, int ch);"
```

String mit einem Zeichen füllen.

Der String wird mit einem bestimmten Zeichen gefüllt.

Aufrufwerte:

-----

---

UBYTE \*spstr - Pointer auf den String der die Zeichenkette erhalten soll.

int num - Länge des zu füllenden Bereiches.

int ch - Ascii-Wert des Charakters mit dem der String gefüllt wird.

Rückgabe:

-----

N/A

## 1.12 instr

"int instr(UBYTE \*sa, UBYTE \*sb);"

Suche nach einem Teilstring im String. Gesucht wird, ob der String 'sb' in 'sa' enthalten ist.

Wird der Suchstring gefunden, dann wird seine Position im String zurückgeliefert, beginnend bei 0.

Ist der Suchstring nicht vorhanden, dann wird -1 zurückgegeben.

Die Schreibweise (Groß-/Kleinbuchstaben) ist unerheblich, es wird intern immer mit Kleinbuchstaben gearbeitet.

Der deutsche Zeichensatz (ä, ö und ü) ist berücksichtigt.

Aufrufwerte:

-----

UBYTE \*sa - String in dem gesucht werden soll.

UBYTE \*sb - Teilstring der gesucht wird.

Rückgabe:

-----

int - Position von sb in sa. Kann 0 bis (strlen(sa)-strlen(sb)) sein.

Enthält -1 wenn nichts gefunden wird.

## 1.13 upstr

"void upstr(UBYTE \*trptr);"

Alle Buchstaben im String in Großbuchstaben wandeln.

Im Gegensatz zu ähnlichen Funktionen wird der deutsche Zeichensatz bearbeitet.

ä -> Ä, ö -> Ö und ü -> Ü.

Aufrufwerte:

-----

UBYTE \*trptr - Pointer auf den String der gewandelt werden soll.

Rückgabe:

-----

N/A

## 1.14 lowstr

```
"void lowstr(UBYTE *trptr);"
```

Alle Buchstaben im String in Kleinbuchstaben wandeln.

Im Gegensatz zu ähnlichen Funktionen wird der deutsche Zeichensatz bearbeitet.

Ä -> ä, Ö -> ö und Ü -> ü.

Aufrufwerte:

-----

UBYTE \*trptr - Pointer auf den String der gewandelt werden soll.

Rückgabe:

-----

N/A

## 1.15 set

```
"void set(UBYTE *lbuff, int slen);"
```

String auf feste Länge bringen. Zu lange Strings werden abgeschnitten, zu kurze rechts mit Leerzeichen gefüllt.

Das ganze geht auch mit einem sprintf, allerdings ist der wesentlich langsamer und braucht zusätzlich einen Pufferstring.

Aufrufwerte:

-----

UBYTE \*lbuff - String der bearbeitet werden soll.

int slen - Länge die der String haben soll.

Rückgabe:

-----

N/A

## 1.16 midstr

```
"void midstr(UBYTE *mstr, int pos, long laenge);"
```

String ausschneiden.

Ab der Position 'pos' (0 bis strlen(mstr)-1) die per 'laenge' angegebene Anzahl Zeichen herausschneiden.

Ist 'laenge' zu groß wird bis zum Stringende herausgeschnitten. 'laenge' -1 bedeutet das gleiche, als ab 'pos' bis zum Ende.

Ist 'pos' ungültig wird nichts bearbeitet.

Aufrufwerte:

-----

UBYTE \*mstr - String der bearbeitet werden soll.

int pos - Position ab der ausgeschnitten wird.

int laenge - Anzahl der auszuschneidenden Zeichen (-1 = Rest des Strings).

Rückgabe:

-----

N/A

---

## 1.17 date\_to\_day

"ULONG date\_to\_day(ULONG date);"

Ermittelt den Tag im Jahr (1-365). Es wird eine Überschlagrechnung für einen sogenannten Cruncher durchgeführt. +/- 2 Tage Abweichung stören da nicht und können auch auftreten.

Aufrufwerte:

-----

ULONG date - Datum im Format ttmjjjj

Rückgabe:

-----

ULONG - Tag im Jahr (1-365)

## 1.18 date\_to\_zahl

"ULONG date\_to\_zahl(UBYTE \*da);"

Datum in Ziffern überführen. Das zugewiesene Datum wird als Ziffernfolge in einem unsigned long zurückgeliefert.

Wird z.B. der 12.12.1990 als String übergeben kommt 12121990 zurück.

Aufrufwerte:

-----

UBYTE \*da - Datum im Format tt.mm.jjjj wobei der Trenner egal ist.

Datum könnte auch tt-mm-jjjj sein. Wichtig ist die das die Ziffern korrekt positioniert sind.

Rückgabe:

-----

ULONG - Datum als Zahl im Format ttmjjjj wobei es hier natürlich keine führenden Nullen im Tag geben kann!

## 1.19 time\_to\_zahl

"ULONG time\_to\_zahl(UBYTE \*ti);"

Zeit in Ziffern überführen. Die zugewiesene Zeit wird als Ziffernfolge in einem unsigned long zurückgeliefert.

Wird z.B. 17:50:00 als String übergeben kommt 175000 zurück.

Falls die Sekunden nicht erwünscht sind müssen sie trotzdem erstmal übergeben werden und dann die Rückgabe durch 100 geteilt werden.

Vorsicht! 00:00:00 liefert 0 zurück, also vorsicht mit Divisionen, immer erst prüfen.

Aufrufwerte:

-----

UBYTE \*ti - Zeit im Format SS:mm:ss wobei der Trenner egal ist.

Zeit könnte auch SS/mm/ss sein. Wichtig ist die das die Ziffern

korrekt positioniert sind.

Rückgabe:

-----

ULONG - Zeit als Zahl im Format SSmmss wobei es hier natürlich keine führenden Nullen in den Stunden geben kann! Die Zeit 00:01:30 liefert 130 als Ergebnis.

## 1.20 kill\_ansi

```
"void kill_ansi(UBYTE *buffer);"
```

Alle Ansi-Sequenzen in einem String entfernen und den reinen Ascii-String zurückliefern.

Diese Routine ist in für den Mailboxeinsatz geschaffen worden und sucht nicht wirklich alle definierten Ansi-Befehle.

Erkannt wird lediglich das ESC am Anfang und dann wird bis zum nächsten Buchstaben einschließlich geschnitten.

Ansi-Sequenzen die nicht mit einem Buchstaben enden führen dazu das zuviel weggeschnitten wird!

Die Routine soll lediglich Cursorbewegungen, Farben und Attribute bereinigen um den String ohne Sonderzeichen auf einem Serverschirm abzubilden.

Aufrufwerte:

-----

UBYTE \*buffer - String der vom Ansi bereinigt werden soll.

Rückgabe:

-----

N/A

## 1.21 newer

```
"BOOL newer(UBYTE *d1, UBYTE *t1, UBYTE *d2, UBYTE *t2);"
```

Vergleicht ob Datum 'd1' und Zeit 't1' NEUER als 'd2' und 't2' sind und gibt dementsprechend TRUE oder FALSE zurück.

Aufrufwerte:

-----

UBYTE \*d1 - 1. Datum in der Form tt.mm.jjjj

UBYTE \*t1 - 1. Zeit in der Form SS:mm:ss

UBYTE \*d2 - 2. Datum in der Form tt.mm.jjjj

UBYTE \*t2 - 2. Zeit in der Form SS:mm:ss

Rückgabe:

-----

BOOL - TRUE wenn das 1. Datenpaar neuer ist als das zweite, sonst FALSE.

---

## 1.22 swapmem

```
"void swapmem(char *src, char *dst, int n);"
```

Speicherbereiche vertauschen.

Aufrufwerte:

-----

char \*src - Pointer auf 1. Speicherbereich.

char \*dst - Pointer auf 2. Speicherbereich.

int n - Anzahl der Bytes die ausgetauscht werden sollen.

Rückgabe:

-----

N/A

## 1.23 memncmp

```
"int memncmp(char *a, char *b, int length);"
```

Speicherbereiche vergleichen.

Aufrufwerte:

-----

char \*a - Pointer auf 1. Speicherbereich.

char \*b - Pointer auf 2. Speicherbereich.

int lenght - Anzahl der Bytes die verglichen werden sollen.

Rückgabe:

-----

int - 0 wenn beide Speicherbereiche gleich sind.

1 wenn a größer als b ist.

-1 wenn a kleiner als b ist.

## 1.24 StrCaseCmp

```
"int StrCaseCmp(char *s1, char *s2);"
```

String s1 und s2 werden auf gleichheit geprüft. Groß-/Kleinschrift wird nicht berücksichtigt, jedoch sehr wohl z.B. in den deutschen Umlauten.

Diese Funktion dient intern zum Vergleich von Laufwerksnamen, daher sind die deutschen Umlaute hier nicht umgewandelt.

Aufrufwerte:

-----

char \*s1 - Pointer auf 1. String.

char \*s2 - Pointer auf 2. String.

Rückgabe:

-----

int - 0 wenn beide Speicherbereiche gleich sind.

1 wenn sie unterschiedlich sind.

---

## 1.25 trim\_includes

```
"void trim_includes(UBYTE *trptr);"
```

Siehe trim.

Hier wird jedoch alles entfernt, was nicht zu einem Include-File-Namen gehören kann, z.B.

auch " am Anfang und Ende (diese Sonderversion wird von Pmm benutzt).

Aufrufwerte:

-----

UBYTE \*trptr - Pointer auf den String der bereinigt wird. Nach dem Funktionsaufruf ist der String bereinigt.

Rückgabe:

-----

N/A

## 1.26 mail\_trim

```
"void trim_includes(UBYTE *trptr, int fkt);"
```

Siehe trim.

Hier wird jedoch alles entfernt, was nicht zu einem eMail-Namen gehören kann, z.B.

auch < > am Anfang und Ende (diese Sonderversion wird von Gateway's eMail-System benutzt).

Aufrufwerte:

-----

UBYTE \*trptr - Pointer auf den String der bereinigt wird. Nach dem Funktionsaufruf ist der String bereinigt.

int fkt - 1, wenn < > entfernt werden sollen

0, wenn < > erhalten werden sollen

Rückgabe:

-----

N/A

## 1.27 newstr

```
"void newstr(UBYTE *istr, UBYE *nstr, int pos, int len);"
```

Diese Funktion ist Teil des Gateway-Interpreters und dient fast ausschließlich zur Ersetzung von Variablen durch ihren Datenwerte. Dabei wird aus istr ab der Stelle pos eine durch len festgelegte Anzahl Zeichen entfernt. An gleicher Stelle wird dafür nstr eingesetzt. nstr kann kürzer, länger oder genau so lang sein, wie der entfernte string. Ist nstr leer (0 Zeichen), dann wird nur der angegebene Bereich entfernt.

Eine Überprüfung der Gültigkeit von pos und len findet nicht statt. Das heißt im Klartext, das fehlerhafte Werte irgendwo im Speicher herumhantieren könnten. Der

Gateway-Interpreter liefert immer verlässliche Werte an und das muß unbedingt auch bei jedem anderen Nutzer sichergestellt werden.

Ach ja, warum ausgerechnet ISTR ? Darum:

istr = Interpretierter String

nstr = Neuer Stringbereich

Aufrufwerte:

-----

UBYTE \*istr - Pointer auf den String der bearbeitet werden soll.

UBYTE \*nstr - String der in istr eingesetzt werden soll.

Wird kein String übergeben wird nur ein Bereich entfernt.

int pos - Position, ab der die Ersetzung beginnt.

int len - Anzahl der Zeichen, die aus istr entfernt werden sollen.

Bei Länge 0 erfolgt nur eine Einfügung.

Rückgabe:

-----

N/A

## 1.28 wordwrp

"int wordwrp(UBYTE \*line, UBYTE \*rest, int len);"

Editor-Sub-Funktion

Es wird eine Textzeile in Line übergeben und die reguläre Maximalbreite der Zeilen in len. Typisch ist ein String mit 80 Zeichen beim einem maximalen Wert von 79. Ist die Länge des Strings größer oder gleich dem len-Wert. dann erfolgt ein Umbruch, wobei nach links gehend ein Leerzeichen gesucht wird. Ab diesem wird dann line gekappt, der Rest des Strings wird in den String 'rest' übertragen und man erhält zwei Strings zurück.

Leerzeichen am Zeilenende werden entfernt, sie zählen also nicht zu Länge hinzu.

Läßt sich 'line' nicht zerlegen (kein Leerzeichen in den letzten 20 Bytes der Maximallänge oder 'line' kürzer als Maximum wird ein leerer Reststring zurückgeliefert und als Rückgabewert der Funktion eine 0.

Bei erfolgreicher Teilung liefert die Funktion eine 1 zurück.

Aufrufwerte:

-----

UBYTE \*line - Textzeile die bei Bedarf umgebrochen werden soll.

UBYTE \*rest - Teilstring der am Ende von Line abgeschnitten wurde, bzw. char Null, wenn erfolglos.

int len - Anzahl Zeichen des Bildschirms bzw. Position des Umbruchs

Rückgabe:

-----

int - 0 = Erfolglos, line wurde nicht verändert

1 = Erfolgreich, Line und Rest angepasst

---

## 1.29 fn\_splitt

```
"void fn_splitt(char *src, char *drive, char *path, char *name, char *ext);"
```

Dient zum zerlegen von Dateinamen in ihre Hauptkomponenten. 'src' kann jede Art von Laufwerks-, Pfad- und/oder Dateinamen sein.

drive:path/name.ext wird aufgespalten in die vier Teilkomponenten. Fehlt ein Part ist sein String leer.

Dabei geht die Funktion davon aus, im Zweifel ist ein Dateiname. Oder einfacher: src ist solange der Name, bis das Gegenteil bewiesen ist. Erst wird nach einem Punkt für eine Namenserverweiterung geschaut, danach ob Slash oder Doppelpunkt zu finden und abzutrennen sind.

Aufrufwerte:

-----

char \*src - String der zerlegt werden soll.

char \*drive - Rückgabe Laufwerksname

char \*path - Rückgabe Pfadnamen

char \*name - Rückgabe Dateiname

char \*ext - Rückgabe Dateiextension

Rückgabe:

-----

char \* - identisch mit src

## 1.30 fn\_build

```
"void fn_build(char *dst, char *drive, char *path, char *name, char *ext);"
```

Fügt die Teilkomponenten drive, path, name und ext zu einem vollständigen Dateinamen zusammen. Drive muß mit einem Doppelpunkt enden. Fehlt am Ende des Pfadnamens der Slash wird angehängt. Beginnt die Extension nicht mit einem Punkt wird dieser ebenfalls zugefügt. Leere Strings bleiben unberücksichtigt. So muß kein Pfad angegeben werden, wenn z.B. die Datei in einem Drive liegt.

Aufrufwerte:

-----

char \*dst - String der aufgebaut werden soll.

char \*drive - Laufwerksname:

char \*path - Pfadnamen/

char \*name - Dateiname

char \*ext - .Extension

Rückgabe:

-----

char \* - Ergebnis (dst) 'drive:path/name.ext'

---

### 1.31 addval

```
"void addval(UBYTE *str, ULONG n);"
```

Zur numerischen Bearbeitung von Zahlenfeldern in Ascii-Dateien.

str wird in ein long gewandelt, n wird aufaddiert und das ganze als 10stelliger String in str wieder zurückgeliefert.

Aufrufwerte:

-----

char \*addval - String mit numerischem Inhalt. Nach Ende der Funktion ist str auf jeden Fall 10

Zeichen groß 'Formatstring ist %10lu'.

ULONG n - Zahl, die auf den numerischen Wert von str aufaddiert wird.

Rückgabe:

-----

N/A

### 1.32 strdup

```
"char *strdup(char *s);"
```

Dupliziert den String \*s in einen neuen Speicherbereich.

Aufrufwerte:

-----

char \*s - Originalstring

Rückgabe:

-----

char \* - Kopie von \*s (Unbedingt daran denken das dies ein malloc ist, also den Speicher am Ende wieder freigeben!

### 1.33 index

```
"char *index(const char *str, int c);"
```

Sucht nach dem ersten Auftreten von c in str und liefert den Pointer auf diese Position zurück.

Aufrufwerte:

-----

const char \*s - Zu durchsuchender String

int c - Ascii-Wert des Zeichen

Rückgabe:

-----

char \* - Pointer auf die Fundstelle von c. Falls nicht erfolgreich wird 0 übergeben.

---