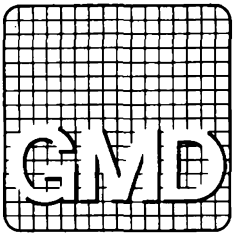
EUMEL

Extendable multi User Microprocessor ELAN-system

Korrekturen und Nachträge I
zum
EUMEL – Benutzerhandbuch 1.7

Stand: 9.5.84



Inhalt

Hinweis:

Diese Dokumentation wurde mit größtmöglicher Sorgfalt erstellt. Dennoch wird für die Korrektheit und Vollständigkeit der gemachten Angaben keine Gewähr übernommen. Bei vermuteten Fehlern der Software oder der Dokumentation bitten wir um baldige Meldung, damit eine Korrektur möglichst rasch erfolgen kann. Anregungen und Kritik sind jederzeit willkommen.

1. Korrekturen

Seite 68

Die letzten drei Sätze unter HOP RETURN sind ersatzlos zu streichen.

(Nach HOP RETURN befindet sich der Cursor in der ersten Zeile des Bildschirms).

Seite 104

Bei dem Beispiel in der Beschreibung der Prozedur 'type' muß der Parameter von 'sqrt' '2.0' und nicht '2' lauten.

Seite 106

Die Beschreibung der Prozedur 'word' ist zu streichen. Dafür ist einzufügen:

TEXT PROC word
Zweck: Liefert das aktuelle Wort (bis zum nächsten Leerzeichen oder Zeilenende).

TEXT PROC word (TEXT CONST sep)
Zweck: Liefert das aktuelle Wort (bis zum nächsten 'sep' - Zeichen oder Zeilenende).

TEXT PROC word (INT CONST len)
Zweck: Liefert das aktuelle Wort mit der Länge 'len'.

Seite 129

Faktoren in der 'type' – Anweisung werden zur Zeit nicht ausgewertet. Deshalb entfällt in dem Abschnitt "Unterschiedliche Schrifttypen: 'type'" von

```
Manche Drucker haben ....
```

bis zum Ende des Abschnitts.

Seite 133

Die Angabe in der Tabelle in der Reihe DIN A4 muß nicht

```
13.5 cm
```

sondern

```
16.0
```

heißen.

Seite 165

Bei Anweisung 'page length' muß der Parameter ein REAL (anstatt INT) sein.

Desgleichen bei den zwei Parametern der Anweisung 'start'.

Seite 273, 322 und 429

Die Prozedur, die den internen Task – Bezeichner der Drucker – Task liefert, heißt nicht

```
print
```

sondern

```
printer
```

Das muß auf folgenden Seiten geändert werden:

Seite 273 dritte und 15. Zeile von oben.
Seite 322
Seite 429 3. – 5. Beispiel

Der Druckerspooler heißt dementsprechend nicht PRINT, sondern PRINTER (ebenfalls Seite 429).

Seite 282

Bei Archiv – Operationen kann es zu folgenden Fehlermeldungen kommen:

* archive read/write impossible
  Bedeutung:
  Archiv – Floppy wurde nicht eingelegt; oder die Tür des Laufwerks ist nicht geschlossen; oder die Floppy ist schreibgeschützt.

* archive read/write error
  Bedeutung:
  Die Floppy ist (teilweise) nicht mehr les – oder beschreibbar.
  Abhilfe:
  Mit 'fetch all (archive)' die noch lesbaren Dateien ins System holen und dann die Floppy neu formatieren. Tritt der Fehler auch dann noch auf, sollte man die Floppy nicht mehr verwenden.

* archive is not yours
  Bedeutung: Archiv wurde nicht angemeldet.
  Abhilfe:
  archive ("name") geben.

* not enough system storage
  Bedeutung:
  Im System ist nicht genügend Platz, um eine datei vom Archiv zu laden.
  Abhilfe:
  Ggf. Dateien im System löschen.

* rerun during archive access
  Bedeutung:
  Das System wurde auf einen 'checkpoint' zurückgesetzt. Zu diesem Zeitpunkt wurde eine Archiv – Operation durchgeführt.
  Abhilfe:
  Archiv – Kommando ggf. erneut geben.

* archive is named ...
  Bedeutung:
  Falscher Archivname eingestellt oder es wurde eine Archivfloppy mit einem anderen Archivnamen eingelegt.
  Abhilfe:
  Archiv – Kommando ggf. erneut geben.

* ... does not exists
  Bedeutung:
  Die Datei "..." gibt es nicht auf dem Archiv.
  Abhilfe:
  Ggf. neuen Dateinamen angeben oder richtige Archivfloppy einlegen.

* ... not found due to directory read error
  Bedeutung:
  Die Datei "..." konnte auf dem Archiv nicht gefunden werden, weil das Inhaltsverzeichnis nicht vollständig gelesen werden konnte.
  Abhilfe: keine.

* save/erase impossible due to directory read error
  Bedeutung:
  Eine datei konnte auf dem Archiv nicht gelöscht oder nicht geschrieben werden, weil das Inhaltsverzeichnis (siehe obige Fehlermeldung) nicht richtig gelesen wurde.
  Abhilfe:
  Ggf. neue Archivfloppy benutzen.

* " – " invalid name
  Bedeutung:
  " – " ist nicht als Dateiname zugelassen.
  Abhilfe:
  Datei umbenennen.

* "..." cannot be saved (archive full)
  Bedeutung:
  Das Archiv ist voll.
  Abhilfe:
  Neue Archivfloppy benutzen.

* archive inconsistent
  Bedeutung:
  Die eingelegte Floppy hat nicht die Struktur einer Archivfloppy (z.B. eine Hintergrund – Floppy oder 'clear (archive)' vergessen).
  Abhilfe:
  Richtige Floppy einlegen.

Wird ein 'list (archive)' gemacht, kann als letzter Eintrag

    follwing entries unreadable in list

erscheinen. Dies bedeutet, daß das Inhaltsverzeichnis nicht vollständig gelesen werden konnte (und damit die dazugehörigen Dateien nicht mehr zugriffsfähig sind).
Abhilfe:
Keine.

Seite 283

Erste bzw. zweite Zeile von oben:

'save' und 'fetch' sind vertauscht.

Seite 332

Das Kommando

```
free (2.0)
```

in der Mitte der Seite ist durch

page

zu ersetzen.

Seite 334

Zeile 8 bzw. 9:

'change' und 'replace' sind vertauscht.

Seite 355

Die 13. Zeile von oben ('0 < from < 255') ist durch

```
length (pattern) < 255
```

zu ersetzen. Die 18. Zeile von oben ('1 < = code (x) < = 254) ist ersatzlos zu streichen.

2. Nachträge

Seite 44 bzw. Seite 47/48

Mit der Informationsprozedur

```
task status
```

können Sie sich über den Zustand einer Task informieren. Beispiele:

```
task status              (* informiert über die eigene
                            Task                          *)
task status (father)     (* informiert über die Vater-Task *)
```

'task status' informiert u.a. über die verbrauchte CPU – Zeit der Task, den belegten Speicherplatz (man beachte, daß Dateien mit Vater – Tasks oder Sohn – Tasks werden), den Kanal, an dem die Task angekoppelt ist und dem Zustand der Task (vergl. auch 'task info').

Mit der Prozedur

```
task info
```

können Sie eine Übersicht über alle in dem System befindlichen Tasks erhalten.

task status

PROC task status

Zweck: Informationsprozedur über den Zustand der eigenen Task. Informiert u.a. über

- Name der Task, Datum und Uhrzeit;
- verbrauchte CPU – Zeit;
- belegten Speicherplatz;
- Kanal, an den die Task angekoppelt ist;
- Zustand der Task (rechnend u.a.m.);
- Priorität.

PROC task status (TASK CONST t)

Zweck: Wie obige Prozedur, aber über die Task mit dem internen Tasknamen 't'. Beispiel:

```
task status (father)
```

task info

PROC task info

Zweck: Informiert über alle Tasknamen im System unter gleichzeitiger Angabe der Vater/Sohn – Beziehungen (Angabe durch Einrückungen).

PROC task info (INT CONST art)

Zweck: Informiert über alle Tasks im System. Mit 'art' kann man die Art der Zusatz – Information auswählen. Für 'art' sind zur Zeit folgende Werte zugelassen:

1: entspricht 'task info' ohne Parameter, d.h. gibt nur die Tasknamen unter angabe der Vater/Sohn – Beziehungen aus.

2: gibt die Tasknamen aus. Zusätzlich erhalten Sie Informationen über die verbrauchte CPU – Zeit der Task, die Priorität, den Kanal, an dem die Task angekoppelt ist und den eigentlichen Taskstatus. Hierbei bedeuten:

0	– busy –	Task ist aktiv.
1	i/o	Task wartet auf Beendigung des Outputs oder auf Eingabe.
2	wait	Task wartet auf Sendung von einer anderen Task.
4	busy – blocked	Task ist rechenwillig, aber blockiert.
5	i/o – blocked	Task wartet auf I/O, ist aber blockiert.
6	wait – blocked	Task wartet auf Sendung, ist aber blockiert. Achtung: Die Task wird beim Eintreffen einer Sendung automatisch entblockiert.

3: wie 2, aber zusätzlich wird der belegte Speicher angezeigt. (Achtung: Prozedur ist aufwendig!). Beachten Sie, daß Dateien mit Väter/Söhnen "geshared" werden. Beispiel:

Mit 'begin ("sohn", "vater")' wird eine neue Task eingerichtet. Für 'sohn' wird in diesem Zustand der gleiche belegte Speicher angezeigt wie für 'vater'. Erst wenn (Datei-) Operationen in der Sohn- oder Vater-Task vorgenommen werden, wird ein unterschiedlicher Speicherplatz angezeigt.

Seite 70

Werden Tabulatormarken gesetzt (HOP TAB), gelten die voreingestellten Tabulatormarken (Anfang und Ende einer Zeile) nicht mehr. Dies ist z.B. bei dem Schreiben von Tabellen notwendig. Andererseits möchte man beim Schreiben von "normalen" Text wieder die voreingestellten Tabulatormarken bedienen können. Mit den Tasten

```
ESC TAB
```

kann man die gesetzten Tabulatormarken (erkenntlich an dem "Dach"-Zeichen in der Kopfzeile) verschwinden lassen. Dann gelten wieder die voreingestellten Marken. Erneutes ESC TAB stellt die gesetzten Tabulatormarken wieder her usw.

Merke: ESC TAB wirkt wie ein Umschalter zwischen voreingestellten und gesetzten Tabulatormarken.

Seite 272

Mit dem Operator

```
/
```

kann man aus einem Tasknamen den internen Tasknamen erhalten. '/' kann überall dort eingesetzt werden, wo ein interner Taskname verlangt wird.

Beispiel:

```
task status (/"meine task")
```

Seite 298

Die Prozedur

```
BOOL PROC pattern found
```

dient dazu, nach einer Suchoperation mit 'up' bzw. 'down' nach dem Erfolg des Suchens zu fragen. Im Gegensatz zu 'at' erspart man sich das nochmalige Schreiben des gesuchten Textes. Beispiel:

```
...
down (file, "muster", 100)
IF pattern found        (* analog at (file, "pattern") *)
  THEN ...
FI
```

Seite 427

Die letzten 5 Zeilen dieser Seite lauten:

PROC note line
Zweck: Zeilenvorschub in der Fehlermeldungsdatei.

note
PROC note (TEXT CONST meldung)
Zweck: Zwischenspeichern von 'meldung'.

Textkosmetik – Makros

Makros dienen als Abkürzung für immer wiederkehrende Textteile und/oder Kommandos.

Textkosmetik – Makros kommen zum Einsatz bei

- immer wiederkehrenden Textteilen;
- immer wiederkehrenden Anweisungssequenzen;
- bei der Erstellung von Manuskripten, deren endgültige Form man anfänglich noch nicht weiß oder die man noch ändern will.

Die Definition von einem oder mehreren Makros wird mit dem Editor vorgenommen. Diese Makro – Datei wird dann geladen. Von diesem Augenblick an "kennen" 'lineform'/'autoform' und 'pageform' die Makros (d.h. die Textzeilen und/oder Anweisungen, die sich unter dem dem Makronamen "verbergen").

'lineform'/'autoform' beachten die Anweisungen, die in den Makros enthalten sind. Man beachte, daß die Anweisungen und Textzeilen, die in den Makros enthalten sind, nicht in der Datei erscheinen. Erst 'pageform' setzt diese in die Druckdatei ein.

Ein Beispiel

Hier wird ein einfaches Beispiel für einen Briefkopf gezeigt.

Angenommen, die Firma 'Meier' schreibt mit dem EUMEL – System ihre Geschäftsbriefe. Sie hat einen Drucker zur Verfügung, mit dem man auch die Briefköpfe erstellen kann. Für den Briefkopf schreibt der Junior – Chef ein Makro 'kopf' in eine Datei 'macro definitionen':

```
#*kopf#
#type("fett und gross")#  Firma Meier
#type("fett")#            Gemischtwaren in kleinen Mengen
#type("klein")#           Straße
                          Stadt
#type ("normal")#
#*macro end#
```

Der Name des Makros ist 'kopf'. Man beachte, daß eine Makro – Definitionmit dem Namen des Makros beginnen müssen. Der Makroname muß mit einem '*' gekennzeichnet werden, um ihn von "normalen" Text – Anweisungen unterscheiden zu können. Jedes Makro wird mit einer 'macro end' – Anweisung beendet (es dürfen mehrere Makros hintereinander in die Datei geschrieben werden).

Nun muß der Junior – Chef das so definierte Makro 'laden':

```
load macros ("macro defintionen")
```

Zur Kontrolle kann er sich die "geladenen" Makros in einen Datei ausgeben lassen:

```
list macros ("kontroll datei")
```

Nun kann der Junior – Chef seiner Sekretärin sagen, daß sie von jetzt ab ein neues Kommando zur Verfügung hat, welches einen Briefkopf vor jeden Brief ausdruckt (mit dem Namen 'kopf'). Die Sekretärin schreibt also nun folgenden Brief:

```
#kopf#

Sehr geehrter Herr ....
usw.
```

Nachdem sie mit 'lineform' den Brief zeilenweise formatiert hat, kontrolliert sie die formatierte Datei. Hier hat sich noch nichts verändert, die neue Anweisung 'kopf' steht immer noch so in der Datei. ('lineform' beachtet zwar alle Anweisungen und Textzeilen, setzt diese jedoch nicht in die Datei ein).

Nun formatiert die Sekretärin die Datei, welche den Brief enthält, mit 'pageform'. In der Druckdatei ist nun die Anweisung 'kopf' verschwunden, dafür stehen aber nun die Zeilen des Makrorumpfes ('pageform' setzt die Zeilen des Makros in die Druckdatei ein):

```
#type("fett und gross")#  Firma Meier
#type("fett")#            Gemischtwaren in kleinen Mengen
#type("klein")#           Straße
                          Stadt
#type ("normal")#


Sehr geehrter Herr ...
usw.
```

Merke: Wie man aus diesem Beispiel sieht, sind Makros bei der Verwendung von wenigen Text – und/oder Anweisungsfolgen nützlich, die immer in der gleichen Form benötigt werden.

Ein Beispiel mit Makro – Parametern

Unserem Junior – Chef fällt nun auf, daß er sein Makro noch etwas verbessern kann. Er will noch das Datum mit in den Briefkopf aufnehmen. Somit editiert er seine Makro – Datei folgendermaßen (man beachte die '$' – Zeichen):

```
#*kopf ($1)#
#type("fett und gross")#  Firma Meier
#type("fett")#            Gemischtwaren in kleinen Mengen
#type("klein")#           Straße
                          Stadtname
#type ("normal")#

                                        Stadtname, den $1
#*macro end#
```

Damit hat er dem 'kopf' – Makro einem Parameter gegeben ('$1'; die Parameter werden numeriert. Ein zweiter Parameter würde '$2' heißen usw.).

Die Sekretärin muß nun die Anweisung 'kopf' mit dem jeweiligen Datum in einen Brief schreiben:

```
#kopf ("9.1.1984")#
```

'pageform' setzt nun das angegebene Datum direkt hinter 'Stadtname, den' in den Briefkopf ein (in der Druckdatei).

Merke: *Durch die Makro – Parameter ist es also möglich, immer wiederkehrende Textteile in Schriftstücke einsetzen zu lassen, die sich nur in Kleinigkeiten unterscheiden.*

Ein Beispiel für Manuskripte

Hier wird gezeigt, wie man mit Makros Anweisungen formulieren kann, die aussagen, um was es sich bei einem Text handelt und nicht, wie es behandelt werden soll.

Bei Manuskripten für Artikel, Bücher und Manuals weiß ein Autor oft vorher nicht, in welchem Format das Manuskript gedruckt werden wird. Zu diesem Zweck ist es ebenfalls nützlich, die Makros zu verwenden. Beispiel:

```
#*kapitel anfang ($1)#
#free (2.0)#
#type ("gross und fett")##ib (9)#$1#ie (9)##type ("normal")#

#*macro end#
```

In diesem Beispiel wird ein Makro für den Anfang eines Kapitels definiert. Zwischen zwei Kapiteln soll hier zwei cm Zwischenraum bleiben, die Kapitel – Überschrift (als Parameter) wird in einer grösseren Schrift gedruckt. Zusätzlich wird die Überschrift in den 9. Index aufgenommen für ein Inhaltsverzeichnis. Nach der Überschrift wird eine Leerzeile eingeschoben, bevor der "richtige" Text anfängt.

Der Anwender dieses Makros schreibt also z.B. folgende Anweisung:

```
#kapitel anfang ("Ein Beispiel fuer Manuskripte")#
```

(Beachte, daß die Kapitel – Überschrift nicht länger als eine Textzeile sein darf. Das liegt daran, das 'lineform'/'autoform' zwar die Zeile bearbeitet, aber nicht in den Text einsetzt. 'pageform' setzt also die unveränderte – nicht aufgebrochene Textzeile – ein).

Man kann nun Makros für die meisten Textstrukturen definieren. Schreibkräfte brauchen dann in der Regel die meisten der Text – Anweisungen nicht zu kennen, sondern nur noch eine Anzahl von einfachen Makro – Anweisungen.

Die Makro – Definitionen können jederzeit geändert werden, um wechselnden Bedürfnissen angepaßt zu werden (z.B. wenn ein Verlag ein bestimmtes Schreibformat verbindlich vorschreibt). In diesem Fall brauchen nicht alle Text – Dateien geändert zu werden, sondern nur die Makro – Definitionen.

Ein weiterer Vorteil einer solchen Vorgehensweise ist, daß die Makro – Anweisungen in diesem Fall angeben, was eine bestimmte Text – Struktur ist, und nicht, wie die Struktur behandelt werden soll.

Anmerkung:
In eine Makro – Definition sollte man ggf. 'limit' –, 'type' – und 'linefeed' – Angaben einsetzen, um die Makros unabhängig von der Aufrufstelle zu machen. Ggf. sollte man die Datei vorher mit 'lineform' bearbeiten, um Trennungen vorzunehmen.

Merke: Makros dienen zur flexiblen Behandlung von Text – Strukturen, indem Makros definiert werden, die angeben, um was es sich dabei handelt.

Beschreibung der Makro – Kommandos

Mit dem Kommando

```
load macros ("macro datei")
```

kann eine Datei, in denen die Makro – Definitionen enthalten sind, in den Makro – Speicher des Textsystems geladen werden. Ist dies fehlerfrei erfolgt, kann man 'lineform'/'autoform' Dateien übergeben, die die definierten Maro – Anweisungen "kennen" und befolgen. 'pageform' setzt bei Antreffen einer Makro – Anweisung den Makrorumpf in die Ausgabe – Datei ein.

Die Definition eines Makros erfolgt mit dem Makronamen, der von Anweisungszeichen eingeschlossen ist. Um Makro – Anweisungen von "normalen" Textkosmetik – Anweisungen zu unterscheiden, müssen diese nach dem ersten Anweisungszeichen mit einem '*' gekennzeichnet werden. Beispiel:

```
#*macro eins#
Makrorumpf mit "normalen" Kommandos, wie z.B.
   #type ("x")#
#*macro end#
```

Der Aufruf eines Makros, welcher z.B. in einer von 'lineform' zu bearbeitenden Datei steht, unterscheidet sich nicht von einer "normalen" Textanweisung. Beispiel:

```
... #macro eins# ...
```

Hat das Makro Parameter (bei der Definition mit '$' – Zeichen durchnumeriert), müssen beim Aufruf TEXT – Parameter eingesetzt werden (also in Anführungsstrichen). Beispiel:

```
#*macro zwei ($1)#
... $1 ...
#*macro end#

(* Aufruf: *)

#macro zwei ("ein einzusetzender Text")#
```

Anmerkung:
Bei Makros gibt es keine generischen Anweisungen. Makros, die gleichen Namen haben, aber sich durch die Anzahl der Parameter unterscheiden, sind also nicht erlaubt.

Übersicht über die Kommandos

list macros

PROC list macros (TEXT CONST datei)

Zweck: Ausgabe der "geladenen" Makros in die Datei 'datei'. 'datei' darf vorher nicht existieren, wird also von 'list macros' eingerichtet. Die "geladenen" Makros bleiben unberührt. Man kann die mit 'list macros' in die Datei 'datei' geschriebenen Makro – Definitionen ggf. verändern und erneut mit 'load macros' laden.

Fehlerfall:

* file already exists
  Ausgabe – Datei 'datei' ist bereits vorhanden.

load macros

PROC load macros (TEXT CONST datei)

Zweck: Lädt Makro – Definitionen in den Makro – Speicher des Textsystems. Die Definitionen müssen in der Datei 'datei' enthalten sein (mit dem Editor erstellen). Es können mehrere Definitionen in der Datei enthalten sein. Um den Makro – Speicher zu leeren, übergibt man eine leere 'datei'.

Eine Makro – Definition besteht aus einem
– Makro – Kopf:
Muß alleine auf einer Zeile stehen. Der Makro – Kopf fängt mit '#*' – Zeichen an und wird mit '#' beendet. Beispiel:

```
#*ein macro#
```

Der Name eines Macros muß (wie alle andern Anweisungen auch) mit kleinen Buchstaben geschrieben werden. Leerzeichen spielen keine Rolle.

Eventuelle Parameter müssen in Klammern (bei mehreren durch Kommata getrennt) und mit einem $ – Zeichen numeriert werden. Beispiel:

```
#*macro1 ($1)#
#*macro 2 ($1, $2)#
```

– Makro – Rumpf:
Besteht aus beliebig vielen Text – Zeilen, die Kommandos enthalten können. Parameter (also das $ – Zeichen mit anschließender Nummer) werden bei Aufruf eines Makros ersetzt. In einem Makro – Rumpf darf keine Makro – Anweisung erscheinen, die noch nicht definiert wurde (sog. "Vorwärts – Referenzen").

– Makro – Ende:
besteht aus der Anweisung

```
#*macro end#
```

und muß wie der Makro – Kopf alleine auf einer Zeile stehen.

Fehlerfälle:

* file does not exists
  Die Eingabe – Datei 'datei' ist nicht vorhanden.
* macro store overflow (number lines)
  Es passen zur Zeit nicht mehr als 1 000 Zeilen in den Makro – Speicher.
* macro store overflow (number macros)
  Es passen zur Zeit nicht mehr als 100 Makro – Defintionen in den Makro – Speicher.

Sortier – Programme

Es stehen zwei verschiedene Sortier – Programme zur Verfügung: 'sort' (Sortierung nach ASCII – Reihenfolge) und 'lex sort' (Sortierung nach deutschem Alphabet).

Das Kommando

```
sort ("datei")
```

sortiert 'datei' zeilenweise. Beispiel:

```
Eingabe-Datei:
   Berta ist eine Frau.
   Adam ist ein Mann.
   ...
Sortierte Datei:
   Adam ist ein Mann.
   Berta ist eine Frau.
   ...
```

Dabei werden die Zeilen – Anfänge solange zeichenweise miteinander verglichen, bis ein Unterschied auftritt und dann ggf. umgeordnet. Werden zwei ungleich lange Zeilen (Anzahl Zeichen/Zeile) miteinander verglichen, dann kann man sich die kürzere Zeile mit Leerzeichen auf die Länge der längeren Zeile verlängert denken.

Die Reihenfolge, in der die Zeilen sortiert werden, erfolgt nach dem ASCII – Zeichensatz in aufsteigender Reihenfolge (vergl. TEIL 3; EUMEL – Zeichencode):

```
Leerzeichen
einige Sonderzeichen
Ziffern
einige Sonderzeichen
Große Buchstaben
einige Sonderzeichen
kleine Buchstaben
einige Sonderzeichen
Umlaute und ß
```

Das bedeutet, daß z.B. folgendermaßen sortiert wird:

```
Adam
Ball
Zuruf
aber das ist ein Satz
niemals
Überlauf
```

Um zu erreichen, daß große und kleine Buchstaben gleichwertig behandelt werden, kann man das Kommando

```
lex sort ("datei")
```

geben. In diesem Fall würde die sortierte Datei folgendermaßen aussehen:

```
aber das ist ein Satz
Adam
Ball
niemals
Überlauf
Zuruf
```

Man beachte, daß der Umlaut 'Ü' wie 'Ue' behandelt wird (für die restlichen Umlaute gilt eine analoge Behandlung; ebenso wird 'ß' wie 'ss' behandelt). Weiterhin werden alle Sonderzeichen bei der Sortierreihenfolge ignoriert.

Sortier – Kommandos

sort

PROC sort (TEXT CONST datei)

Zweck: Die Prozedur 'sort' sortiert die Datei 'datei' zeilenweise. Die Sortierung erfolgt nach der Ordnung, die der EUMEL – Zeichencode vorschreibt. Beispielsweise werden Zeilen ("Sätze"), die mit Ziffern beginnen, vor Sätzen, die mit Buchstaben anfangen, eingeordnet. Sätze, die mit großen Buchstaben beginnen, werden vor Sätzen mit kleinen Buchstaben einsortiert. Weiterhin werden die Umlaute und das "ß" nach allen anderen Buchstaben eingeordnet.

PROC sort (TEXT CONST datei, INT CONST anfang)

Zweck: Sortiert eine Datei wie obige Prozedur, jedoch wird bei der Sortierung nicht der Anfang eines Satzes beachtet, sondern die Position 'anfang'.

lex sort

PROC lex sort (TEXT CONST datei)

Zweck: Wie 'sort', jedoch nach (deutscher) lexikographischer Reihenfolge nach DIN 5007. Bei den Vergleichen werden die Operatoren LEXEQUAL, LEXGREATER, LEXGREATEREQUAL (vergl. TEIL 11 des Benutzerhandbuchs) verwandt. Anmerkung: 'lex sort' ist um einiges langsamer als 'sort'.

PROC lex sort (TEXT CONST datei, INT CONST anfang)

Zweck: Wie 'lex sort', jedoch wird bei der Sortierung bei 'anfang' jeder Zeile begonnen.

Rechnen im Editor: 'tecal' – Kommando

Das Programm 'tecal' (Abkürzung für "text calculator") ermöglicht das einfache Rechnen im EUMEL – Editor.

Das Programm 'tecal' ermöglicht einfache Rechnungen (ähnlich wie mit einem Taschenrechner) unter der Benutzung des Editors. Gleichzeitig stehen dem Benutzer aber alle Fähigkeiten des Editors zur Verfügung. Damit ist es möglich, z.B. Rechnungen auf einfache Weise zu erstellen oder Tabellenspalten zu berechnen.

'tecal' wird wie der Editor mit dem Kommando

```
tecal ("zu editierende datei")
```

aktiviert. (Anmerkung: 'tecal' ist nicht standardmäßig insertiert). Der Benutzer erhält auf dem Bildschirm das gewohnte Editor – Fenster. Nur die unterste Zeile des Bildschirms enthält eine Informationszeile von 'tecal', in der die (Zwischen –) Ergebnisse einer Rechnung zur Kontrolle festgehalten werden.

Merke: 'tecal' ermöglicht einfache Rechnungen im EUMEL – Editor.

Ein einfaches Beispiel

Angenommen, Prokurist Meier der Firma 'Software Experts' muß eine Rechnung schreiben. Er schreibt (nachdem er 'tecal ("Rechnung")' aufgerufen hat) u.a.:

```
...
Wir berechnen Ihnen

1 Manual  'Software Auswahl leicht gemacht'      112.30 DM
1 Manual  'Ohne Fehler programmieren'            300.-

                                          Summe
```

Zuerst löscht Prokurist Meier eventuell vorhandene Zwischenergebnisse von 'tecal' (nach Aufruf von 'tecal' ist dieser immer auf '0.0' gesetzt) mit

```
ESC c
```

Das funktioniert wie eine CLEAR – Taste bei einem Taschenrechner, löscht also ggf. vorhandene Werte. In der Informationszeile (die letzte Zeile des Bildschirms) erscheint darum als Wert '0.0'.

Nun "fährt" er mit dem Cursor auf den ersten Wert ('112.30'). Dabei ist es belanglos, welche Ziffer er "trifft". Dann betätigt er

```
ESC z
```

Damit erscheint dieser Wert in der Informationszeile. Durch 'ESC z' wird versucht, einen Wert aus der Datei zu lesen, die durch den Cursor angezeigt wird. (Gelingt dies nicht, erfolgt in der unteren Zeile eine Fehlermeldung). Dann betätigt er

```
ESC +
```

weil er ja die zwei Werte addieren will. In der 'tecal' – Informationszeile erscheint dazu das Zeichen '+' und das Ergebnis bleibt unverändert. Dann fährt er auf den zweiten Wert und betätigt erneut 'ESC z'. Nun erscheint der zweite Wert als Ergebnis. Um das Ergebnis der Rechnung zu erfahren, betätigt er

```
ESC =
```

Die Summe der zwei Zahlen erscheint nun in der Informationszeile. Nun fährt er mit dem Cursor auf die Stelle, an der die Summe stehen soll und betätigt hier

```
ESC s
```

(für schreiben). Damit erscheint die eben errechnete Summe (412.30) an dieser Stelle der Datei.

Man bedient 'tecal' also wie einen Taschenrechner. Man muß allerdings, um die Rechen-Tasten zu bedienen, ESC zuvor drücken. Dies ist notwendig, um die "normalen" Tasten von den 'tecal'-Tasten zu unterscheiden.

Merke: *Mit einigen einfachen Tastendrücken können Berechnungen vorgenommen werden. 'ESC z' liest einen Wert von der aktuellen Cursor-Position, 'ESC s' schreibt den angezeigten 'tecal'-Wert an die aktuelle Cursor-Position. 'ESC c' löscht alle Werte im 'tecal'-Rechner.*

Einige weitere einfache Rechenoperationen

In diesem Abschnitt werden weitere einfache Operationen von 'tecal' beschrieben.

Natürlich kann man mit 'tecal' nicht nur Addieren. Die folgenden Operationen laufen analog 'ESC +':

```
ESC -          (Subtrahieren)
ESC *          (Multiplizieren)
ESC /          (Dividieren)
```

Beispiel:

```
...
Wir berechnen Ihnen

Artikelbezeichnung          Anzahl  Einzelpreis Summe

Schraube, verdreht            27        1.05     28.35

                                              Gesamt 28.35
```

Dazu drückt Prokurist Meier folgende Tasten:

Cursor auf	Taste	Angabe in 'tecal'-Informations-zeile	
27	ESC c	Clear	0.00
unveraendert	ESC z	Eingelesen	27.00
unveraendert	ESC *	*	27.00
1.05	ESC z	Eingelesen	1.05
unveraendert	ESC =	=	28.35
unter Summe	ESC s	Ausgegeben	28.35
in Gesamt-Zeile	ESC s	Ausgegeben	28.35

Wie wir sehen, kann Prokurist Meier jederzeit seine Eingaben kontrollieren mit Hilfe der 'tecal' – Informationszeile.

Praktischer Tip:

'ESC s' schreibt den aktuellen Wert wie der Dezimal – Tabulator des Editors (siehe EUMEL – Benutzerhandbuch). Die Stelle, an der der Cursor steht, wird beim Schreiben die Stelle, wo der Dezimalpunkt stehen wird. Ziffern vor dem Dezimalpunkt werden also nach links, Ziffern nach dem Dezimalpunkt nach rechts geschrieben.

Es empfiehlt sich also, den Tabulator zu verwenden!

Merke: ESC mit den Tasten '–', '+', '*' und '/' haben die gewohnte Wirkung.

Die Verwendung von Klammern

'tecal' erlaubt bei Rechnungen die Eingabe von Klammern.

Beispiel (wir haben hier die Taste ESC fortgelassen):

```
2 * (3 + 5) = 16.00
```

Merke: Klammern können bei Rechnungen beliebig verwendet werden.

Der Prozent – Operator

Angenommen, wir wollen 14 Prozent von 200 DM errechnen. Dann können wir wie gewohnt verfahren (für bessere Lesbarkeit zeigen wir hier für 'ESC z' den jeweiligen Wert):

```
200 ESC % 14 ESC =
```

Der Prozent – Operator arbeitet also wie die anderen Operatoren auch. Was müssen wir machen, um die 14 Prozent von 200 auf den Wert von 200 zu addieren? Ganz einfach:

```
200 ESC % 14 ESC = ESC + 200 ESC =
```

Wie wir solche Tastensequenzen einfacher erledigen können, zeigen wir im übernächsten Abschnitt.

Merke: Der Prozent – Operator arbeitet wie die anderen Operatoren.

Direkte Eingabe

Es kann ein Wert in 'tecal' direkt eingegeben und gleichzeitig in die Datei geschrieben werden.

Durch das Betätigen von

```
ESC e
```

erscheint 'Direkt' in der 'tecal' – Informationszeile. Nun kann ein Wert (wie im Editor) eingegeben werden, der zuerst nur in der Informationszeile erscheint. Betätigt man RETURN, ist der Wert aufgenommen und wird dann an die aktuelle Cursor – Position in die Datei geschrieben. Man erspart sich also in solchen Fällen ESC z.

Merke: Mit ESC e wird ein Wert direkt in 'tecal' aufgenommen und in die Datei geschrieben.

'tecal' und Lernen im Editor

> *Bei sich wiederholenden Rechnungen ist es sinnvoll, Rechenoperationen "zu lernen" und eine Operation auf eine Taste zu legen.*

Angenommen, Prokurist Meier hat häufig Rechnungen zu schreiben und muß des öfteren die Mehrwertsteuer eines Betrages errechnen. Zu diesem Zweck kann er die "Lern" – Einrichtung des Editors benutzen (vergl. EUMEL – Benutzerhandbuch; Editor – Teil). Beim Lernen "merkt" sich der Editor jeden Tastendruck (also auch 'tecal' – Operationen). Die gelernten Tasten kann man anschließend mit einem Tastendruck abrufen. Meier kann die Operationen wie folgt vom Editor lernen lassen:

Er fährt mit dem Cursor zuerst auf den Wert, von der die Mehrwertsteuer errechnet werden soll. Dann betätigt er ESC HOP (es erscheint LEARN in der Kopfzeile des Editors). Dann schreibt er die 'tecal' – Operationen, wie oben gezeigt (ohne das abschließende ESC s). Mit dem abschließenden ESC HOP und einer weiteren Taste (sagen wir mal 'm' als Abkürzung für Mehrwert) beendet er das Lernen. Nun kann er jederzeit die Mehrwertsteuer eines Betrages errechnen, in dem er ESC m betätigt. Das Ergebnis kann er dann mit ESC s an eine gewünschte Stelle schreiben.

Praktischer Tip:

Tabulator-Bewegungen kann man ebenfalls lernen. So ist es z.B. möglich, die Berechnung von Spalten- oder Reihensummen zu erlernen, indem man mit TAB jeweils zu dem nächsten Wert springt.

Merke: *Es können beliebige Rechnungen erlernt und auf eine Taste gelegt werden. Die gelernten Rechnungen können mit Hilfe einer Taste abgerufen werden.*

Einstellen von Nachkommastellen

Betätigt man

```
ESC n
```

erscheint in der 'tecal'-Informationszeile 'Nachkommastellen ?'. Nun kann man die Anzahl der Nachkommastellen (0-9) einstellen (intern wird aber jeweils mit höchster Genauigkeit gerechnet. Zur Information erscheint in der 'tecal'-Informationszeile immer die eingestellte Anzahl von Nachkommastellen.

Merke: Mit ESN n stellt man die Anzahl der Nachkommastellen ein.

Autor:

Rainer Hahn

Umschlaggestaltung:

Hannelotte Wecken

Druck:

Zentrale Vervielfältigungsstelle der Universität Bielefeld
im Februar 1984