Systemhandbuch

Stand: 08.08.86

Herausgeber:

Gesellschaft für Mathematik und Datenverarbeitung mbH

Postfach 1240, Schloß Birlinghoven
D – 5205 Sankt Augustin 1
Telefon(02241) 14 – 1, Telex 8 89 469 gmd d
Telefax(02241) 14 28 89, BTX *43900#
Teletex 2627 – 224135 = GMDVV

Autoren:

Jochen Liedtke, Dietmar Heinrichs, Rainer Hahn

Texterstellung:

Dieser Text wurde mit der EUMEL – Textverarbeitung erstellt, aufbereitet und mit dem Agfa Laserdrucksystem P400 gedruckt.

Umschlaggestaltung:

Hannelotte Wecken

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.

Inhaltsverzeichnis

Zuerst lesen!

Der größte Teil dieses Systemhandbuchs ist für Anwender geschrieben, die tiefer in das EUMEL – System einsteigen und evtl. Systemergänzungen oder Systemänderungen programmieren wollen. Der erste Teil ist allerdings für alle interessant, die ein EUMEL – System verwenden, selbst für Anfänger, die ihr System zum ersten Mal in Betrieb nehmen wollen. Entsprechend der verschiedenen Adressatenkreise unterscheiden sich die einzelnen Kapitel stark in der Beschreibungsart. Deshalb:

Sind Sie EUMEL – Neuling?

Dann sollten Sie ***vor*** dem Einschalten Ihres Systems die Einführung des Kapitels "System einrichten" lesen. Dort werden keine weiteren Kenntnisse vorausgesetzt. Danach sollten Sie erst einmal durch praktisches Arbeiten mit Hilfe des Benutzerhandbuchs etwas mit dem System vertraut werden.

Haben Sie schon einige Zeit mit dem EUMEL gearbeitet?
Sind Sie mit dem System einigermaßen vertraut?

Dann lesen Sie den kompletten Teil 1 ("System einrichten") dieses Systemhandbuchs. Aber nicht mehr!
Das Lesen der folgenden Kapitel ist für den einfachen Betrieb des EUMEL – Systems nicht erforderlich. Sie setzen auch intime Kenntnis des Systems auf dem Niveau des Benutzerhandbuchs voraus und würden Anfänger leicht verwirren.

Haben Sie Probleme mit Ihrer Hardware?

Wenn Sie nichts von Hardware verstehen, fragen Sie einen Fachmann!

Wenn Sie ein gewisses Grundwissen über Hardware haben, dann lesen Sie Teil 2 ("Hardware und ihre Steuerung"). In diesem Kapitel sollten Sie "3. Kanäle und Konfigurierung" erst einmal auslassen.

Wollen Sie tiefer in das Betriebssystem einsteigen?
Haben Sie EUMEL – Erfahrung?
Haben Sie Programmiererfahrung?

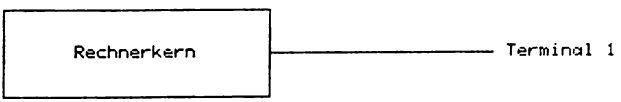
Dann lesen Sie im Systemhandbuch alles, was Ihnen interessant erscheint.

Teil 1: System einrichten

1. Einführung

Wie Ihr System aufgebaut ist

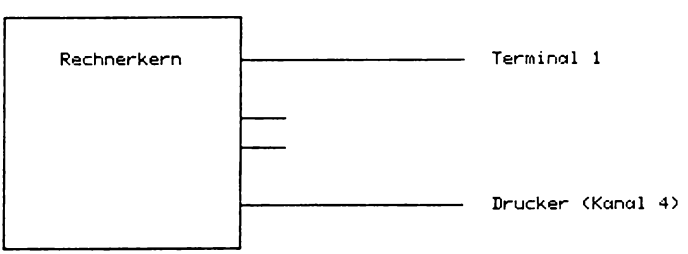
Der kleinstmögliche EUMEL-Rechner besteht aus einem Rechnerkern und einem Terminal:



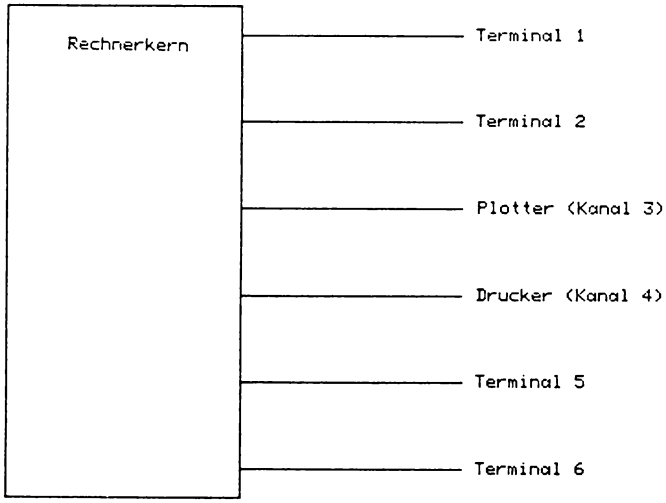
Anmerkung: In manchen Fällen ist das Terminal hardwaremäßig in den Rechner integriert. Trotzdem fassen wir diese physische Einheit dann als zwei logisch getrennte Komponenten auf, nämlich Rechnerkern und Terminal!

Wie man sieht, hat das Terminal die Nummer 1. Das bedeutet, daß es über Kanal 1 mit dem Rechnerkern verbunden ist. Das EUMEL-System kennt 16 solche Kanäle, wobei es von der konkreten Hardware abhängt, welche Kanäle wirklich vorhanden sind und welche Geräte man daran anschließen kann. (Allerdings ist der Kanal 1 als Verbindung zum Terminal 1 immer vorhanden.)

In den meisten Fällen wird auch ein Drucker angeschlossen sein. Die genaue Anschlußart ist wieder von der konkret verwendeten Hardware abhängig. Nehmen wir an, er sei an Kanal 4 angeschlossen:



Man sieht also, daß Lücken bei der Verwendung der Kanäle auftreten dürfen. Bei Multi – User – Systemen können weitere Terminals und andere Geräte (z.B. Plotter) angeschlossen werden:



Wie Sie die EUMEL – Software erhalten und einspielen

Genaue Anweisungen zur Generierung eines EUMEL – Systems können wir hier nicht geben, denn diese hängen von Ihrem Rechnertyp ab. Danach fragen Sie bitte Ihren Hard – und Softwarelieferanten.

Im allgemeinen werden Sie unter anderem Hintergrunddisketten mit der Aufschrift 'HG 0...n' (Hintergrund) und Disketten mit den Aufschriften 'std.devices', 'std.graphik', 'std.printer' und 'std zusatz' (Standardarchive) bekommen haben. Dabei befinden sich auf den Hintergrund – Disketten die standardmäßig immer vorhandenen Teile des EUMEL – Systems, während die Standardarchive viele nützliche Programme enthalten, auf die Sie bei Bedarf zurückgreifen können.

In Ihrem Rechner gibt es höchstwahrscheinlich ein Hintergrund – und ein Archivgerät. Das letztere ist fast immer ein Floppylaufwerk. Als Hintergrundgerät steht meistens ein Plattenlaufwerk oder ein oder mehrere Floppylaufwerke zur Verfügung (genaueres vom Hardwarelieferanten).

Beim ersten Systemstart müssen Sie (nach evtl. notwendigen Generierungsläufen) den Hintergrund auf das Hintergrundgerät kopieren. Dazu gehen Sie folgendermaßen vor:

Zuerst legen Sie die Diskette mit der Aufschrift "HG 0" in das Archivlaufwerk. Dann starten Sie das System. Beim Systemstart führt das EUMEL – System zuerst einen Speichertest durch. Dabei erscheint das Wort "Speichertest:" auf dem Bildschirm des Hauptterminals. Während des Tests werden "*" – Zeichen ausgegeben. In dieser Zeit drücken Sie eine Taste. Dann wird nicht "durchgestartet", sondern es erscheint ein Startmenü, das Ihnen unter anderem

(1) Systemstart
(2) Hintergrund vom Archiv laden
(3) Hardwaretest
(4) neuen Urlader vom Archiv laden

anbietet. Dann (Eingaben von Ihnen sind weiß auf schwarz gedruckt):

2

alten HG ueberschreiben (j/n) j

Jetzt wird der Hintergrund eingespielt. Wenn der Hintergrund auf mehreren Disketten ausgeliefert worden ist, müssen Sie, wenn der EUMEL es verlangt, die nächste Hintergrunddiskette einlegen.
Wenn der gesammte Hintergrund gelesen worden ist, erscheint

"fertig, bitte RESET"

Nehmen Sie jetzt die Diskette aus dem Archivlaufwerk. Nach dem RESET wird das System wieder gestartet (s.o.).

Anmerkung: Auch wenn Sie ein Floppylaufwerk als Hintergrundgerät verwenden, legen Sie den mitgelieferten Hintergrund nie in dieses Laufwerk, um Zerstörungen dieser "Mutter" zu verhindern.

Wie Sie die Konfiguration einstellen

Die verschiedenen auf dem Markt befindlichen Terminal – und Druckertypen sind leider nicht so genormt, daß sie alle gleichartig vom EUMEL – System angesprochen werden können. Deshalb muß man das EUMEL – System bei der Inbetriebnahme ***konfigurieren.***

Die Konfiguration wird automatisch nach dem ersten Systemstart im Dialog erfragt. Alle einstellbaren Gerätetypen werden auf dem Bildschirm aufgelistet. Danach können Sie den Gerätetyp und in vielen Fällen weitere Kenndaten (Baudrate, Bits pro Zeichen, Parität und Stopbits einstellen. Die einzustellenden Werte hängen von den Geräten ab, die Sie an die entsprechenden Kanäle anschließen wollen. Bei Fragen wenden Sie sich deshalb bitte an Ihren Hard – /Softwarelieferanten.
Hier sehen Sie als Beispiel einen Konfigurationsdialog (mit reduzierter Anzahl von Gerätetypen):

psi	transparent	PC.ascii	PC.german
PC.KB2	IBM.PC.AT	M24	TA.P50/60
Siemens.PC – D	FT10/20.ascii	FT10/20.german	M20.original
M20	VIDEOSTAR	Qume.german	VC404.german
VC404.ascii	VC404.hrz	ELBIT.german	ELBIT.ascii
REGENT25	REGENT40	ws580	TVI914.ascii
TVI.german	DM5	GT100	DEC.VT220.ascii
DEC.VT220.german			

Kanal 1 (j/n) j
Kanal 1: transparent (j/n) n
Kanal 1: PC.KB2 (j/n) j

Kanal 2 (j/n) j
Kanal 2: transparent n
Kanal 2: PC.ascii n
Kanal 2: FT10/20.german j
Kanal 2: FT10/20.german, 9600 Baud (j/n) j
Kanal 2: FT10/20.german, 9600, 7 Bits j
Kanal 2: FT10/20.german, 9600, 7, even parity j
Kanal 2: FT10/20.german, 9600, 7, even, 1 Stopbits (j/n) j
Kanal 2: FT10/20.german, 9600, 7, even, 1, ohne Protokoll (j/n) j

Kanal 3 (j/n) j
Kanal 3: transparent (j/n) j
Kanal 3: transparent, ohne Protokoll (j/n) n
Kanal 3: transparent, XON/XOFF – Protokoll (j/n) j
Kanal 3: transparent, XON/XOFF – Protokoll, normale Puffer (j/n) j

Koennen unbenutzte Geraetetypen geloescht werden (j/n) j

Dabei ist folgendes zu beachten:

- Welches Gerät bei Ihnen an welchem Kanal angeschlossen ist bzw. werden kann, erfahren Sie von Ihrem Hard – /Softwarelieferanten.

- Enden die Typbezeichnungen mit "ascii", kennen die Geräte keine Umlaute und kein "ß", aber die Zeichen "[]{}\|". Fehlt eine Typbezeichnung oder ist sie 'german', dann ist der deutsche Zeichensatz mit Umlauten und "ß" vorhanden, evtl. sind aber die Zeichen"[]{}\|" nicht darstellbar. Manche Geräte kann man hardwaremäßig auf "ascii" oder "deutsch" schalten. Üblicherweise wird man dann hardwaremäßig die deutsche Version wählen und auch bei der Konfigurierung einstellen.

- Die angebotenen Codetabellen sind teilweise mit Bezug auf bestimmte Geräte benannt. Das schließt nicht aus, daß sie auch für Ihr Gerät passen, auch wenn dieses nicht aufgeführt ist. Fragen Sie danach den Lieferanten Ihrer Hard – und Software.

- Beim Systemstart befindet sich für jeden Gerätetyp eine Tabelle im System. Um Platz zu sparen, werden die nicht benötigten Tabellen nach der Konfigurierung gelöscht, wenn man auf die Frage "Koennen unbenutzte Geraetetypen geloescht werden (j/n)?" mit "j" antwortet. (Keine Angst, man kann sie von den mitgelieferten Standardarchiven wieder laden.)

Ihr System ist jetzt konfiguriert, soweit etwaige Codeumsetzungen auf den verschiedenen Kanälen betroffen sind. Allerdings fehlen noch die Programme, um einen Drucker zu betreiben. Lesen Sie dazu Kapitel 2, sobald Sie mit Ihrem System etwas vertrauter geworden sind.
Im übrigen ist Ihr System jetzt fertig, und Sie können wie in der Beispielsitzung (Kap. 1.2) des Benutzerhandbuchs fortfahren. Im Multi – User – Betrieb müssen Sie das System durch Drücken der SV – Taste anrufen, um ein Supervisor – Kommando geben zu können.

2. Ausführliche Beschreibung

System laden

Wie in der Einführung dieses Kapitels beschrieben ist, geht man beim Systemstart durch Eingabe eines Zeichens während des Vortests in das Startmenü und wählt dort "Hintergrund vom Archiv laden" an. Falls der zu ladende Hintergrund sich über mehrere Archiv – Disketten erstreckt, werden die folgenden sukzessive angefordert.

System sichern

Der aktuelle eigene Hintergrund läßt sich (mit allen Tasks und allen Dateien) durch das Kommando

save system

auf Archivdisketten sichern. Dabei wird erst ein 'shutup' durchgeführt. Anschließend werden *formatierte* Disketten angefordert. Der Hintergrund wird komprimiert gesichert, d.h. nur die belegten Blöcke werden auf das Archiv geschrieben.

Anmerkung: Diese Prozedur kann in Multi – User – Systemen nur von privilegierten Tasks (Nachfahren von "SYSUR"), wie dem OPERATOR, aufgerufen werden.
Vor dem Aufruf von 'save system' sollten Sie genügend Disketten formatiert haben (Überprüfen Sie mit 'storage info', wieviel Disketten Sie benötigen, um den gesammten Hintergrund darauf zu schreiben).

Multi – User – System gegen Unbefugte schützen

Falls der Benutzerkreis eines Multi – User – Systems nicht "gutartig" ist, sollte man verhindern, daß jeder Benutzer des Systems Zugang zu privilegierten Operationen hat, wie Löschen anderer Tasks, Konfiguration ändern und System sichern.

Dies erreichen Sie dadurch, daß Sie **alle** privilegierten Tasks, das sind 'SYSUR' und alle Söhne, Enkel usw. von 'SYSUR', durch Paßworte schützen. Damit wird der Zugang zu diesen Tasks nur möglich, wenn man das entsprechende Paßwort eingibt. Man definiert solche ***Task – Paßworte***, indem man die zu schützende Task mit Hilfe des Supervisor – Kommandos "continue" an ein Terminal holt und dann das Kommando

```
task password ("simsalabim")
```

gibt. Dabei ist "simsalabim" nur ein Beispiel. Bitte verwenden Sie ein anderes Paßwort! Da die Eigenschaft, privilegiert zu sein, nur davon abhängt, im "SYSUR" – Zweig (und nicht im normalen "UR" – Zweig) des Systems zu sein, könnte sich ein gewitzter Anwender die Privilegierung einfach erschleichen, indem er eine neue Sohntask von "SYSUR" einrichtet. Um auch diese Möglichkeit zu unterbinden, sollte man in **jeder** Task des SYSUR – Zweiges ebenfalls ein ***"begin" – Paßwort*** definieren. Das geschieht mit dem Kommando

```
begin password ("simsalabim")
```

Bei der Wahl der Paßworte sollte man folgendes bedenken:

- Ein zu kurzes oder offensichtliches Paßwort (beispielsweise der Name des Systemverwalters) kommt schnell heraus.

- Oft werden Paßworte bekannt, weil irgendwo ein Zettel mit den Paßworten herumliegt.

- Der Paßwortschutz ist hart. Wenn man sein Paßwort vergessen hat, gibt es keinen Zugang mehr zu der geschützten Task.

Beschreibung der Paßwortprozeduren:

task password

PROC task password (TEXT CONST password)

Zweck: Einstellen eines Paßwortes für eine Task im Monitor.

begin password

PROC begin password (TEXT CONST password)

Zweck: Verhindert das unberechtigte Einrichten einer Sohn – Task.

Anmerkung: Das 'begin password' vererbt sich auf die später erzeugten Sohn – Tasks.

info password

PROC info password (TEXT CONST old, new)

Zweck: Mit dieser Prozedur kann das Info – Paßwort (max. 9 Zeichen) gesetzt bzw. geändert werden.

BEACHTE: Anschließend wird automatisch ein Shutup durchgeführt.

family password

PROC family password (TEXT CONST password)

Zweck: Setzt oder ändert das Paßwort derjenigen Familienmitglieder, die kein Paßwort oder das gleiche Paßwort wie die aufrufende Task haben.

Zu einer Familie gehören die Task in der man sich befindet und die ihr untergeordneten Tasks.

Bsp.: Das Kommando 'family password ("EUMEL")' wird in SYSUR gegeben. Dadurch wird das SYSUR – Paßwort und die Paßworte der entsprechenden Tasks unter SYSUR auf "EUMEL" gesetzt.

Konfiguration im Multi – User – System

In Multi – User – Systemen läuft die Konfiguration über die Task "configurator" ab. Diese Task müssen Sie also für die hier aufgeführten Operationen durch das Supervisor – Kommando "continue" ankoppeln. (Dabei wird das Paßwort überprüft, falls die Task geschützt wurde.)

Anmerkung: Man kann die Task "configurator" löschen und dann neu (als Sohn, Enkel,... von SYSUR) wieder einrichten. Danach holt man die Konfigurationsdatei (z.B. von std.devices) und gibt das Kommando "configuration manager".

Der in der Einführung unter "Wie Sie die Konfiguration einstellen" beschriebene Konfigurationsdialog läßt sich vermittels des Kommandos

configurate

aufrufen. Dabei wird für jeden angewählten Kanal die bis jetzt gültige Einstellung als Vorschlag mit ausgegeben. Die Einstellung aller Kanäle, die nicht angesprochen werden, bleibt unverändert.

Im Menü werden die Namen aller Dateien mit Gerätetabellen aufgeführt, die in der Task enthalten sind. Daraus folgt, daß nur noch die bei der letzten Konfigurierung benutzten Typen aufgeführt werden, wenn vorher auf die Frage "Koennen unbenutzte Geraetetypen geloescht werden (j/n)?" mit "j" geantwortet wurde. Nun sind alle ausgelieferten Gerätetabellen auf dem Archiv "std.devices" vorhanden, so daß man wieder benötigte Gerätetypen dadurch verfügbar machen kann, daß man die entsprechenden Dateien von dem Archiv lädt. Dieses Archiv enthält auch die Generatoren der Gerätetabellen (z.B. "FT10/20.german.gen"), die es erlauben, die Tabellen abzuändern. Das sind ELAN – Programme, die die Tabellen erzeugen. Näheres dazu findet man in "Teil 2, 3. Kanäle und Konfigurierung".

Im Konfigurationsdialog kann folgendes eingestellt werden:

Typ — Es werden alle vorhandenen Gerätetabellen durchgegangen, bis eine davon ausgewählt wurde. Diese manchmal etwas langwierige Arbeit kann man durch Eingabe des Zeichens ESC abkürzen: Danach kann man den Typnamen direkt eingeben. *Das funktioniert aber nur vernünftig, wenn das eigene Arbeitsterminal bereits richtig konfiguriert worden ist!*

Baudrate — (nur für V.24 – Kanäle von Bedeutung) Es werden alle einstellbaren Baudraten durchgegangen, bis eine davon ausgewählt wurde. Das sind maximal die Werte 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 19200, 38400 Baud. Meistens kann man aber nur eine Untermenge davon einstellen.

Bits — (nur für V.24 – Kanäle von Bedeutung) Es werden alle einstellbaren Zeichengrößen durchgegangen. In der Regel sind nur 7 oder 8 Bit pro Zeichen von Bedeutung.

Parität — (nur für V.24 – Kanäle von Bedeutung) Möglich sind die Einstellungen 'no', 'even' und 'odd'.

Stopbits — (nur für V.24 – Kanäle von Bedeutung) Stopbits geben die Pause zwischen zwei aufeinanderfolgenden Zeichen an. Möglich sind 1, 1.5 oder 2 Stopbits.

Protokoll — Terminals u.ä. werden üblicherweise ohne Protokoll angeschlossen. Bei langsamen Geräten wie Druckern bzw. Plottern oder aber bei Rechnerkopplungen bzw. Netzen kann der Empfänger nicht immer so schnell Zeichen annehmen wie sie von der Gegenstation gesendet werden. In diesem Fall kann man das XON/XOFF – oder das RTS/CTS – Protokoll einstellen.
BEACHTE: Sender und Empfänger müssen auf das gleiche Protokoll eingestellt sein.

Manchmal müssen auch Terminals mit Protokoll angeschlossen werden. Üblicherweise wählt man dann aber ein rein ausgabeseitiges Protokoll, damit SV den EUMEL auf jeden Fall erreicht.
Es gibt folgende Protokolle:

XON/XOFF – Protokoll:
: Rechner und Gerät steuern die Sendungen jeweils über XON/XOFF – Zeichen.

RTS/CTS – Protokoll:
: Rechner und Gerät steuern ihre Sendungen jeweils über RTS/CTS – Leitungen.

XON/XOFF – ausgabeseitig:
: Das angeschlossene Gerät steuert die Ausgabe über XON/XOFF.Eingaben zum Rechner unterliegen keinem Protokoll.

RTS/CTS – ausgabeseitig:
: Das angeschlossene Gerät steuert die Ausgabe über RTS/CTS. Eingaben zum Rechner unterliegen keinem Protokoll.

XON/XOFF – eingabeseitig:
: Der EUMEL – Rechner steuert die angeschlossenen Geräte durch XON/XOFF. Die Ausgaben zum Gerät unterliegen keinem Protokoll.

RTS/CTS – eingabeseitig:
: Der EUMEL – Rechner steuert die angeschlossenen Geräte durch RTS/CTS. Die Ausgaben zum Gerät unterliegen keinem Protokoll.

Puffer
: Terminals und alle Ausgabegeräte (Drucker u.ä.) haben standardmäßig die normalen "kleinen" Eingabepuffer im System zugeordnet. Bei Rechner – Rechner – Kopplungen, DFÜ oder Netzen kann ein "großer" Eingabepuffer von 512 Byte notwendig werden. Dementsprechend sind Großpuffer nur beim Schnittstellentyp 'transparent' möglich.

Im Konfigurationsdialog werden bei jedem Kanal nur die dort vorhandenen Möglichkeiten angeboten. Dabei wird die vorherige Einstellung immer als erste angeboten. So kann man sich verhältnismäßig einfach "durchtasten".

Die Fragen des Konfigurationsdialogs werden nach folgendem Schema gestellt:

```
erfrage ("Kanal") ;
erfrage ("Typ") ;
IF dieses ist ein v24 kanal
  THEN IF baudrate einstellbar
         THEN erfrage ("Baudrate")
       FI ;
       IF zeichengroesse einstellbar
         THEN erfrage ("Bits")
       FI ;
       IF parität einstellbar
         THEN erfrage ("Parität")
       FI ;
       IF stopbits einstellbar
         THEN erfrage ("Stopbits")
       FI ;
FI ;
erfrage ("Protokoll") ;
IF typ ist tranparent
  THEN erfrage ("Puffer")
FI.
```

Will man seine eingestellte Konfiguration sichern, reicht es, alle Dateien der Task "configurator" auf ein Archiv zu schreiben. Diese Konfiguration kann man dann bei einem neuen Hintergrund einfach vom Archiv laden. Um die Konfigurierung dann auch auszuführen, gibt man das Kommando "setup".

Konfiguration im Single – User – System

Der Konfigurationsdialog erfolgt genauso, wie im vorigen Abschnitt ("Konfiguration im Multi – User – System") beschrieben.

Die Initialisierung des Systems gemäß der eingestellten Konfiguration erfolgt automatisch bei jedem Systemstart nach ordnungsgemäßem "shutup". Im Gegensatz zum Multi – User muß die Initialisierung der internen Gerätetabellen bei einem ***RERUN – Start*** (d.h. das System wurde vorher nicht korrekt mit "shutup" abgeschaltet) im Single – User explizit durch das Kommando

setup

vorgenommen werden.

Anmerkung: Gerade in RERUN – Situationen ist man aber oft nicht im Monitor oder Editor. Um ein laufendes Programm abzubrechen, braucht man dann u.U. die SV – Taste. Diese ist aber gerätespezifisch und steht erst zur Verfügung, nachdem die Gerätetabellen geladen worden sind. Zur Abhilfe wirkt <CONTROL g> (Control – und g – Taste gleichzeitig) als SV – Taste, solange die Gerätetabellen noch nicht geladen worden sind.

Deshalb: Falls notwendig das laufende Programm durch <CONTROL g> abbrechen und dann bei "gib kommando:" das Kommando "setup" geben.

Druckersoftware im Multi – User einrichten

Das Standardarchive "std.printer" enthält einige Druckeranpassungen für die Ansteuerung diverser Druckertypen. Soll einer dieser Druckertypen an das EUMEL – System angeschlossen werden, so muß zuerst eine Task "PRINTER" als Sohntask von "SYSUR" mit dem Supervisorkommando

```
begin ("PRINTER", "SYSUR")
```

eingerichtet werden. In dieser Task müssen dann die folgenden Schritte vollzogen werden:

- Anmelden des Archivs:
  ```
  archive ("std.printer")
  ```

- Holen der Druckeranpassung vom Archiv:
  ```
  fetch ("printer.druckertyp", archive)
  ```

- Ausschalten der Zeilennummergenerierung bei der Insertierung:
  ```
  check off
  ```

- Insertieren der Druckeranpassung:
  ```
  insert ("printer.druckertyp")
  ```

Beispiel:

```
archive ("std.printer")
fetch ("printer.epson.fx", archive);
check off;
insert ("printer.epson.fx")
```

Nach der Insertierung wird zuerst nach dem Druckerkanal gefragt. Dieser sollte mit der Gerätetabelle 'transparent' konfiguriert sein. Dann werden druckerspezifische Fragen zur Papierbreite, Positionierungsart oder ähnlichem gestellt, die mit 'j' oder 'n' beantwortet werden müssen. Dabei werden alle Alternativantworten zu der jeweiligen Frage hintereinander angeboten, bis eine Alternative mit 'j' beantwortet wird. Schließlich muß nochmals das Archiv mit der entsprechenden Fonttabelle eingelegt werden, um diese zu laden. Wenn die Generierung beendet ist, muß im Multi – User Betrieb in allen bestehenden Tasks – insbesondere in der Task 'PUBLIC' – die

Fonttabelle mit dem fonttable – Kommando eingestellt werden. Mit dem Kommando

```
print ("dateiname")
```

wird dann eine Datei ausgedruckt.

Befindet sich keine passende Druckeranpassung für den anzuschließenden Druckertyp auf dem Standardarchiv "std.printer", so sollte die Druckeranpassung "printer.std" benutzt werden. Diese Druckeranpassung ist eine universelle Druckeranpassung für alle Drucker, die mit ASCII – Code 13 ein 'Carriage Return' (d.h. Bewegung des Druckkopfes an den linken Rand) und mit ASCII – Code 10 eine Zeilenschaltung von 1/6 Zoll vornehmen. Mit ihr kann dann in einem Schrifttyp (entweder 10 oder 12 Zeichen pro Zoll, je nachdem welche Fonttabelle eingestellt ist) gedruckt werden. So erhält man wenigstens eine Minimalansteuerung des Druckers. Für eine bessere Ansteuerung des Drucker muß ein Programm geschrieben werden, das das Druckertreiber – Interface erfüllt (siehe Teil 6 "Der EUMEL – Drucker") und eine Fonttabelle erstellt (siehe Teil 7 "Der Fontspeicher") werden.

Druckersoftware im Single – User einrichten

Die Installation der Druckersoftware im Single – User erfolgt ähnlich wie die im Multi – User. Hier brauchen nur die Schritte durchgeführt zu werden, die im Multi – User in der Task "PRINTER" druchgeführt werden müssen. Eine Task "PRINTER" braucht nicht eingerichtet zu werden.

Teil 2: Hardware und ihre Steuerung

Vorwort

Die Hardware eines jeden EUMEL – Systems läßt sich in Rechnerkern und Peripherie einteilen.

a) Der Rechnerkern

In der Regel wird der Rechnerkern aus folgenden Komponenten bestehen:

- CPU
- Vordergrundspeicher (oft als RAM bezeichnet)
- Hintergrundspeicher (Floppy, Harddisk, oder auch RAM/ROM)

Alle Daten, Dateien und Programme werden auf dem Hintergrundspeicher abgelegt. Der benötigte Platz wird dabei dynamisch nach Bedarf zugewiesen. Jeder Zugriff auf Daten, die sich auf dem Hintergrundspeicher befinden, muß über den Vordergrundspeicher erfolgen. Zu diesem Zweck verlagert das EUMEL – System automatisch alle aktuell benötigten Daten in den Vordergrundspeicher. Das erfolgt nach dem Prinzip des Demand – Paging (s. Benutzerhandbuch Kap. 1). Die CPU führt die aktiven Programme (unter Benutzung des Speichers) aus. Dabei bearbeitet sie reihum alle rechenwilligen Prozesse.
Die drei Komponenten des Rechnerkerns werden vollständig vom EUMEL – Betriebssystem verwaltet und miteinander verknüpft, so daß der Anwender sich in der Regel darum weder kümmern muß noch kann. Ausgenommen davon sind allerdings die Diagnose von Hardwarefehlern und Überlegungen zur Systemleistung.

b) Die Peripherie

Alle anderen Geräte oder Gerätekomponenten gehören aus der Sicht des EUMEL – Systems zur Peripherie. Wesentliches Kennzeichen ist, daß sie über Kanäle mit dem Rechnerkern verbunden sind und von dort aus durch System – und Anwenderprogramm gesteuert werden können. Angeschlossen werden können u.a.

- Terminals
- Drucker und Plotter
- andere Rechner bzw. Rechnernetze
- Archivgeräte (z.B. Floppy – Laufwerke)

In der Regel hat jedes EUMEL – System mindestens ein Terminal und Archivlaufwerk. Auch wenn dieses "Terminal 1" und das Floppy – Laufwerk baulich in den Rechner integiert sind, gehören sie logisch zur Peripherie. Die entsprechenden Kanäle sind dann allerdings Teil des Rechners und brauchen den Anwender nicht zu interessieren. Die beiden wesentlichen anderen Kanaltypen sind:

- serielle Schnittstellen (V.24)
- Parallelschnittstellen

Beide führen "echt" aus dem Rechner heraus und sind u.U. hardwaremäßig für den Anwender von Bedeutung. Normalerweise sollte zwar der Lieferant der EUMEL – Hardware für die Verkabelung und den Anschluß peripherer Geräte sorgen, aber Kenntnisse können in Fehlersituationen (z.B. Kabelbruch), bei Umkonfigurierungen und bei Kombinationen verschiedener Geräte helfen.

1. Hardware – Test

Der EUMEL – Hardware – Test ist ein rechnerunabhängiger Test und kann demzufolge nicht so viel überprüfen wie Testprogramme, die genau auf eine entsprechende Hardware zugeschnitten sind. Trotzdem sollten die meisten Hardware – Fehler schon mit dem EUMEL – Hardware – Test gefunden werden.

Bei jedem Systemstart wird der "Vortest" durchgeführt. Nachdem er Terminals, Speicher und Hintergrund angezeigt hat, testet er einmal den Hauptspeicher. Danach wird das eigentliche EUMEL – System gestartet.

Durch Eingabe eines beliebigen Zeichens während des Vortests kommt man in den ausführlichen Start – Dialog. Dort wird u.a. auch die Möglichkeit "Hardware – Test" angeboten. Wählt man diese an, werden die verfügbaren Tests als Menü aufgelistet. Bei jedem EUMEL – System stehen mindestens folgende Testmöglichkeiten zur Verfügung:

(1) Speichertest
(2) Kanaltest
(3) Hintergrundtest
(4) Archivtest

Alle Tests sind dabei Dauertests, d.h. sie beginnen nach jedem Durchlauf von neuem, können aber durch <ESC> abgebrochen werden.

Rechnerabhängig können weitere Tests angeboten werden.

Im folgenden sind aber nur die 4 Standardtests näher beschrieben.

Speichertest

Der Speichertest soll den Vordergrundspeicher (RAM) des Rechners untersuchen. Gerade Speicherfehler tendieren aber dazu, nur sporadisch aufzutreten oder wärmeabhängig zu sein. Deshalb sollte der Test bei Verdacht auf Speicherfehler längere Zeit (einige Stunden) laufen. Leider können auch dann nicht alle Fehler aufgedeckt werden, z.B. nicht solche, die nur in ganz speziellen Situationen entstehen, wie Speicherzugriff mit gleichzeitig anlaufendem Floppymotor und Zeichenausgabe.
Generell gilt hier (wie für jeden Test), daß die Abwesenheit von Fehlern nie Vollkommen sicher nachgewiesen werden kann.

Der Speichertest teilt den Speicher in drei verschiedene Bereiche auf:

```
0  :  adresse MOD 3 = 0
1  :  adresse MOD 3 = 1
2  :  adresse MOD 3 = 2
```

Der freie Speicher wird nach folgendem Algorithmus geprüft:

```
schreibe  (1, OLOLOLOL) ;  out ("**") ;
schreibe  (2, OLOLOLOL) ;  out ("**") ;
schreibe  (0, LOLOLOLO) ;  out ("**") ;
pruefe    (1, OLOLOLOL) ;  out ("**") ;
schreibe  (1, LOLOLOLO) ;  out ("**") ;
pruefe    (2, OLOLOLOL) ;  out ("**") ;
pruefe    (0, LOLOLOLO) ;  out ("**") ;
pruefe    (1, LOLOLOLO) ;  out ("**") ;
schreibe  (0, OLOLOLOL) ;  out ("**") ;
pruefe    (0, OLOLOLOL) ;  out ("**") ;
schreibe  (2, LOLOLOLO) ;  out ("**") ;
pruefe    (2, LOLOLOLO) ;  out ("**") .
```

Dabei werden durch 'PROC schreibe (INT CONST bereich, BYTE CONST muster)' alle Bytes des entsprechenden Bereichs mit dem angegebenen Muster geladen. 'PROC pruefe (INT CONST bereich, BYTE CONST soll)' überprüft entsprechend alle Bytes des Bereichs darauf, ob sie das Sollmuster enthalten.

Findet der Speichertest Fehler, können u.a. folgende Ursachen vorliegen:

- Ein Speicherchip ist defekt.
- Die Versorgungsspannung für den Speicher (meistens +5V) ist zu niedrig, d.h. das Netzteil ist nicht richtig eingestellt bzw. defekt. (Das kann insbesondere dann entstehen, wenn ein Rechner so "hochgerüstet" wurde, daß das Netzteil nachgeregelt werden müßte.)
- Die Kontakte der Speicherkarten sind locker oder oxidiert.
- Die Speicheransteuerung ist defekt.

Kanaltest

Beim Kanaltest werden andauernd auf allen Terminalkanälen (außer auf Terminal 1) die jeweiligen Kanalnummern in der Form "Kanal: n" ausgegeben. Jedes Eingabezeichen wird in dezimaler Verschlüssung unter Angabe der Kanalnummer auf dem Terminal 1 gemeldet.

Mit Hilfe dieses Tests können u.a. Kabel und Geräteeinstellungen überprüft werden. Mögliche Fehlerursachen:

- falsche Baudrate eingestellt

  Symptome: Bei Aus- und Eingabe werden vollkommen unsinnige Zeichen angeliefert.
  Abhilfe: Baudrate am Endgerät oder am Rechner richtig einstellen.

- falsche Parität eingestellt

  Symptome: Einige Zeichen werden richtig übertragen, andere verfälscht. In einigen Fällen können auch alle Zeichen falsch übertragen werden.
  Abhilfe: Parität am Endgerät oder am Rechner richtig einstellen.

- falsches Kabel (z.B. Sende – und Empfangsleitungen fälschlich gekreuzt bzw. nicht gekreuzt, Kabel ohne Flußkontrolle an Schnittstelle mit Flußkontrolle, V.24 – Kabel an Parallelschnittstelle oder umgekehrt):

  Symptome: Keine Ausgabe, keine Eingabe oder andauernder Strom von "Schrottzeichen".
  Abhilfe: richtiges Kabel nehmen oder Kabel korrigieren.

- defektes Kabel (Kabelbruch, defekter Stecker o.ä.)

  Symptome: beliebig.
  Testmöglichkeit: Kabel wechseln.

- defektes Endgerät

  Symptome: beliebig.
  Testmöglichkeit: Anderes Gerät mit gleicher Einstellung (Baudrate, Parität usw.) anschließen.

- defekte Schnittstelle im Rechner

  Symptome: beliebig
  Testmöglichkeit: Endgerät mit gleichem Kabel an eine andere Schnittstelle am Rechner anschließen (dazu evtl. die Geräteparameter wie Baudrate anpassen).

Hinweis: Die Kanäle sind nicht konfiguriert. Fragen Sie Ihren Hard – /Softwarelieferanten nach der Voreinstellung von Kanälen.

Hintergrundtest

Zur Überprüfung des Hintergrundes werden drei Tests angeboten:

(1) Lesetest
(2) Lese – /Schreibtest
(3) Positioniertest

Der ***Lesetest*** prüft, ob alle für EUMEL verfügbaren Blöcke auf der Platte bzw. Floppy lesbar sind. Dabei wird der Blockinhalt nicht inspiziert. Sowohl behebbare (soft) als auch harte Lesefehler werden gemeldet. Der Bediener kann einen Korrekturversuch durch Rückschreiben veranlassen. Bei einem Softerror (Block konnte nach mehreren Versuchen doch gelesen werden) wird der gelesene Block neu geschrieben. Der Fehler kann jetzt ohne negative Folgen behoben sein, bei defekter Hardware aber auch zu Folgefehlern führen.
Als Korrekturversuch bei harten Fehlern wird ein mit 'FFFD' gefüllter Block geschrieben. Wird ein solcher Block später vom EUMEL gelesen und als Code angesehen, führt das zur Fehlermeldung "code block unreadable". Wird FFFD als INT angesehen, liefert es den Wert –3, bei REAL oder TEXT können keine Vorhersagen gemacht werden.

Bei dem ***Lese – /Schreibtest*** wird jeder Block mit mehreren Bitmustern beschrieben und zur Kontrolle wieder gelesen. Der alte Inhalt wird vor dem Test gesichert und nachher wieder in den Block geschrieben.

Achtung: Normalerweise zerstört der Test den EUMEL – Hintergrund nicht. Bei defekter Platte können allerdings Blöcke durch mißlungenes Rückschreiben zerstört werden.

Der ***Positioniertest*** arbeitet ähnlich wie die Leseprüfung. Allerdings wird in der Reihenfolge 0, 1, 0, 2, 0, 3, ... gelesen, so daß die Platte für jeden Lesevorgang positionieren muß.

Achtung: Wegen der harten Plattenbelastung sollte dieser Test nicht zu lange laufen.

Archivtest

Der Archivtest arbeitet ähnlich wie der Hintergrundtest – allerdings auf dem Archiv. Er kann sowohl zur Überprüfung von Archiv – Datenträgern (Lesetest) als auch zum Test des Archivlaufwerks benutzt werden.

2. Serielle Geräteschnittstelle

Pinbelegung und Kabel

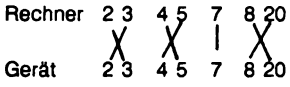
Anmerkung: Dieses Kapitel ist nur für solche Anwender von Bedeutung, die sich selbst mit der Verkabelung ihrer Geräte befassen.

Im folgenden werden die wichtigsten Leitungen der offiziellen V.24 – Schnittstelle (serielle Schnittstelle zum Anschluß von Terminals, Druckern, Fremdrechnern u.ä.) beschrieben:

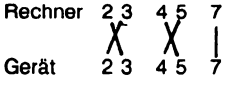
Pin	Betriebsrichtung	Bedeutung
2	out	Sendedaten
3	in	Empfangsdaten
4	out	Sendeaufforderung (RTS)
5	in	Empfangsbereitschaft (CTS)
7		Signalerde
8	in	Gegenstation bereit (DCD)
20	out	eigene Station bereit (DTR)

Dabei dient das Paar (2,3) zur Übertragung der Daten, mit Hilfe von (4,5) ist Flußkontrolle möglich (z.B. kann ein Drucker damit Sendungen vom Rechner "verlangsamen"). Das Paar (8,20) wird bei manchen Geräten und Rechnern benutzt, um festzustellen, ob die Gegenstation eingeschaltet ist.

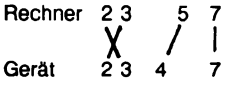
Die meisten Rechner haben die gleiche Pinbelegung wie oben aufgeführt. Die Kabel müssen dann die folgenden Pins verbinden:



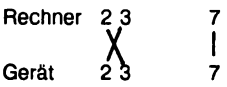
Vollständige Verbindung mit Flußkontrolle.



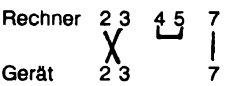
Reicht für die meisten Anschlüsse mit Flußkontrolle, z.B. Rechnerkopplung.



Reicht für die meisten Drucker, Flußkontrolle nur einseitig vom Drucker zum Rechner.

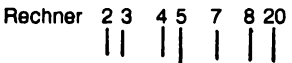


Reicht meistens für Terminals, Flußkontrolle ist dabei überflüssig.

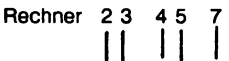


Manchmal für Terminals. Rechnerseitig wird Flußkontrolle durch die Brücke 4 – 5 simuliert.

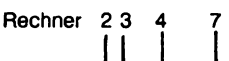
Bei manchen Rechnern werden die notwendigen paarweisen Vertauschungen schon im Rechner durchgeführt. Es ergibt sich entsprechend:



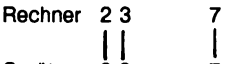
Vollständige Verbindung mit Flußkontrolle.



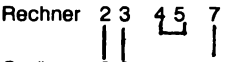
Einfacher Anschluß mit Flußkontrolle.



Drucker, einseitige Flußkontrolle.



Terminal.



Terminal mit simulierter Flußkontrolle.

3. Kanäle und Konfigurierung

Im EUMEL – System dienen Kanäle zur Kommunikation mit der Außenwelt, d.h. Kanäle sind Verbindungen vom Rechner zu peripheren Geräten wie Terminals, Drucker, Plotter und Archiv. Kanäle können für zeichen – und blockorientierte Ein – /Ausgabe verwendet werden. Im Multi – User – System heißt ein Kanal privilegiert, wenn er nur von privilegierten Systemtasks (Nachkommen des Supervisors) benutzt werden kann.

Kanalaufteilung:

Kanal	Bedeutung
1	zeichenorientiert, blockorientiert Dieser Kanal muß mit einem Terminal verbunden sein, da über ihn der Systemstart erfolgt.
2 – 15	zeichenorientiert, blockorientiert Diese Kanäle werden für weitere Terminals, Drucker, Plotter, Rechnerkopplung usw. verwandt.
16 – 23	blockorientiert
24 – 30	blockorientiert, privilegiert
31	blockorientiert, privilegiert Dieser Kanal ist der Standardkanal des Archivsystems, d.h. üblicherweise wird darüber die Archivfloppy angesprochen.
32	blockorientiert, privilegiert Dieses ist ein interner Kanal, an den kein externes Gerät angeschlossen werden kann. Er wird zur Konfigurierung der anderen Kanäle benutzt.

Der Supervisor des EUMEL – Systems verwaltet die Kanäle. Jeder Task ist dabei kein oder genau ein Kanal zugeordnet. Entsprechend ist jedem Kanal keine oder genau eine Task zugeordnet. Solche Zuordnungen können von außen durch den Benutzer über die SV – Kommandos 'continue' und 'break' (nur bei interaktiven Kanälen) und vom Programm selbst über die Prozeduren 'break' und 'continue' (s. Teil 5) verändert werden. In jedem Fall überprüft der Supervisor die Zugriffsberechtigung.

Zeichenorientierte Ein – /Ausgabe

Zeichenorientierte Ein – /Ausgabe kann auf den Kanälen 1 bis 15 benutzt werden. Dafür stehen die Basisoperationen

```
PROC out (TEXT CONST text)
PROC outsubtext (TEXT CONST source,
                 INT CNST from)
PROC outsubtext (TEXT CONST source,
                 INT CONST from, to)
PROC cursor (INT CONST x, y)
PROC inchar (TEXT VAR char)
TEXT PROC incharety
TEXT PROC incharety (INT CONST time limit)
PROC get cursor (INT VAR x, y)
```

und alle darauf aufbauenden Operationen (wie 'put', 'get', 'putline', 'getline' usw.) zur Verfügung. Diese Kanäle sind 'konfigurierbar' (s.u.) und erlauben den Anruf des Systems durch den Benutzer von außen (SV – Taste). In der Regel werden die Kanäle 1 bis 15 für Terminals, Drucker, Plotter und andere zeichenorientierte Anschlüsse benutzt.
Wenn ein Kanal zum Anschluß eines Terminals verwendet wird, müssen die Standard – Steuerzeichen des EUMEL – Systems (s. Benutzerhandbuch, Teil 3 "Editor", "5. EUMEL – Zeichensatz") auf jedem Terminal die gleiche Semantik haben. Das heißt beispielsweise, daß der Code ""2"" auf jedem Terminal bei Ausgabe den Cursor um eine Stelle nach rechts verschiebt. Da Datenendgeräte in dieser Hinsicht aber faktisch keiner Norm gehorchen, müssen die EUMEL – Codes in der Regel in terminalspezifische Codes umgesetzt werden. Diese Umsetzregeln kann man bei der Konfigurierung (s.u.) festlegen. Für die meisten Terminaltypen werden allerdings fertige Konfigurationssätze mit dem EUMEL – System zusammen ausgeliefert, die man bei der Einrichtung des Systems (s. Systemhandbuch Teil 1) interaktiv anwählen kann.

Blockorientierte Ein – /Ausgabe

Blockorientierte Ein – /Ausgabe kann auf den Kanälen 1 bis 32 benutzt werden. Dafür stehen die Operationen

```
PROC control (INT CONST code1, code2, code3,
                    INT VAR return code)
PROC blockout (DATASPACE CONST ds,
                    INT CONST page nr, code1, code2, INT VAR return code)
PROC blockout (ROW 256 INT CONST block,
                    INT CONST code1, code2, INT VAR return code)
PROC blockin (DATASPACE VAR ds,
                    INT CONST page nr, code1, code2, INT VAR return code)
PROC blockin (ROW 256 INT VAR block,
                    INT CONST code1, code2, INT VAR return code)
```

zur Verfügung. Näheres findet man in Teil 4.5 dieses Systemhandbuchs.

Konfigurierung von Kanal 1 bis 15

Alle zeichenorientierten Kanäle können (vermittels Block I/O auf Kanal 32) konfiguriert werden. Dabei werden im wesentlichen Umsetzregeln für Ein- und Ausgabe definiert, die den Zweck haben,

- bei der Ausgabe den EUMEL Zeichensatz auf den Zeichensatz des angeschlossenen Geräts abzubilden und

- bei der Eingabe die gerätespezifischen Zeichen auf den EUMEL Zeichensatz abzubilden.

So ist eine geräteunabhängige Programmierung möglich.

Mit Hilfe der Prozedur 'link' kann man einen der Kanäle 1 bis 15 auf einen bestimmten Typ setzen. Immer vorhanden sind die Typen:

"transparent": Keine Codeumsetzungen (für Drucker usw.) und
"psi" : Keine Codeumsetzungen, jedoch folgende Sonderfunktionen:

Code		Funktion
7	(CTLg)	SV
17	(CTLq)	Stop
23	(CTLw)	Weiter
4	(CTLd)	Info

Weitere Typen müssen in Form eines DATASPACE gleichen Namens in der Task vorliegen, in der das Kommando 'link' gegeben wird. Einige Typen werden auf Archiv mit ausgeliefert. (s. Systemhandbuch Teil 1).

Neue Terminaltypen können mit den Prozeduren 'new type', 'enter outcode', 'enter incode' usw. definiert werden. Im einzelnen stehen folgende Prozeduren zur Verfügung:

link

PROC link (INT CONST channel, TEXT CONST type)

Zweck: Der angegebene Kanal (1 bis 16) wird auf den angegebenen Typ konfiguriert.

Hinweis: Die Prozedur 'link' hat die angegebene Wirkung nur, wenn die Task an Kanal 32 hängt, der nur für Söhne des SUPERVISOR zugänglich ist ('continue (32)').

y size

PROC y size (INT CONST channel, new size, INT VAR old size)

Zweck: Einstellmöglichkeiten für verschiedene Bildschirmgrößen. Diese Prozedur wirkt nur auf Kanal 32. 'channel' gibt dabei den zu konfigurierenden Kanal an.
z.B für 25 Zeichen, wie beim PC – D.

new type

PROC new type (TEXT CONST typ)

Zweck: Eröffnet einen neuen Kanaltyp mit dem Namen 'typ'. Die folgenden Aufrufe von 'enter outcode', 'enter incode' usw. beziehen sich dann auf diesen Typ.

enter outcode

PROC enter outcode (INT CONST eumelcode, zielcode)

Zweck: Legt fest, daß der Code 'eumelcode' bei Ausgabe auf dem Terminaltyp in 'zielcode' gewandelt werden soll.

PROC enter outcode (INT CONST eumelcode, TEXT CONST zeichen)

Zweck: Wirkt wie 'enter outcode (eumelcode, code (zeichen))'.

PROC enter outcode (INT CONST eumelcode, zeit, TEXT CONST seq)

Zweck: Hiermit wird festgelegt, daß der Code 'eumelcode' als Mehrzeichenfolge 'seq' ausgegeben werden soll. Jedesmal, wenn diese Folge ausgegeben wurde, verzögert das System die Ausgabe des nächsten Zeichens um mindestens 'zeit' Millisekunden. Dies wird z.B. von den meisten Terminals gefordert, wenn sie die Funktion 'Löschen Bildschirm' ausführen sollen.

enter incode

PROC enter incode (INT CONST eumelcode, TEXT CONST seq)

Zweck: Es wird festgelegt, daß eine Eingabezeichenfolge 'seq' an das System als ein (!) Zeichen mit dem Code 'eumelcode' weitergegeben werden soll. Die ganze Sequenz muß dabei innerhalb von ca. 40 Millisekunden eintreffen, andernfalls werden die Zeichen einzeln gemeldet. Diese Logik ist erforderlich, um auch Terminals anzuschließen, die z.B. Cursortasten als ESC – Sequenzen melden. Ohne die Zeitüberwachung würde das Betätigen der ESC – Taste sonst die Eingabe blockieren, bis die Folge 'seq' vollständig ist.

Folgende Eumelcodes sind für die Sondertasten (SV usw.) anzugeben:

17	STOP
23	WEITER
4	INFO
7	SV

Weitere Codes ('HOP',...) sind im Benutzerhandbuch Version 1.7 Seite 111 angegeben.

Hinweis: Liefert die SV – Taste eines Terminals von sich aus schon Code 7, so ist dennoch 'enter incode (7, ""7"")' anzugeben. Entsprechendes gilt für die anderen drei "Ereignistasten" STOP, WEITER und INFO. Bei allen anderen Tasten brauchen jedoch nur echte Umcodierungen vermerkt zu werden.

cursor logic

PROC cursor logic (INT CONST offset, modus, TEXT CONST pre, mid, post)

Zweck: Es wird festgelegt, daß der EUMEL – Code 6 (Cursorposition) mit den folgenden beiden Zeichen, deren Codes y und x seien,

bei modus = 255 als
    pre + code (offset + y) + mid + code (offset + x) + post
und bei modus = 1 als
    pre + text (offset + y) + mid + text (offset + x) + post

ausgegeben wird.

Hinweis: 'offset' ist üblicherweise 32 (manchmal 0) und
mid = post = "".

cursor logic

PROC cursor logic (INT CONST dist, TEXT CONST pre, mid, post)

Zweck: Diese Prozedur wird von den Konfigurationsdateien alter Versionen benutzt.

ansi cursor

PROC ansi cursor (TEXT CONST pre, mid, post)

Zweck: Diese Prozedur ist anstelle von 'cursor logic' zu verwenden, wenn die Cursor – Positionierungen bei dem Terminal so erfolgt, wie im Ansi – Standard definiert wird.

elbit cursor

PROC elbit cursor

Zweck: Diese Prozedur ist bei Elbit – Terminals anstelle von 'cursor logic' zu verwenden.

Beispiele für die Verwendung von 'new type', 'enter outcode', 'enter incode', 'cursor logic' , 'ansi cursor' und 'elbit cursor' findet man in den "...gen" Dateien auf dem Archiv "std.devices".

Konfigurations – Manager

Wenn das System gestartet wird, weiß der Urlader noch nicht, welche Terminaltypen an welchen Kanälen hängen. (Der Vortest kann deshalb auch nicht bildschirmorientiert arbeiten).

Falls eine Task 'configurator' im System ist, schickt der SUPERVISOR dieser eine Startsendung zu. Diese Task kann daraufhin die nötigen Konfigurierkommandos ('link',...) ausführen.

Ansonsten ist 'configurator' ein normaler Fontmanager, der die Fonttabellen verwaltet (siehe Teil 7). Deshalb sollte im System immer eine Task 'configurator' existieren und nach Möglichkeit immer im 'wait' stehen. Man kann ihn also auch mit 'continue' an ein Terminal holen und dann wie üblich Kommandos geben.

configurate

PROC configurate

Zweck: Führt den Konfigurationsdialog und anschließendes 'setup' durch.

setup

PROC setup

Zweck: Alle Kanäle werden gemäß der im letzten Konfigurationsdialog bestimmten Werte konfiguriert (das wird automatisch bei jedem Systemstart durchgeführt – im Single – User – System allerdings nur bei normalen Starts, nicht im Rerun).

configuration manager

PROC configuration manager

Zweck: Durch Aufruf dieser Prozedur wird die Task zu einem Konfigurationsmanager. Man kann also die Task "configurator" löschen, neu als Systemtask einrichten und mit diesem Kommando wieder etablieren.

BEACHTE: Die Task muß 'configurator' heißen.

Hinweis: Bei einem Multi – User kann es passieren, daß eine Task schon Ausgaben macht, bevor der Kanal konfiguriert ist (z.B. wenn ein 'shutup' bei aktiver Netz – Kommunikation durchgeführt wurde).

Teil 3: ELAN – Programme

1. Wertebereiche und Speicherbedarf

INT – Objekte

Jedes Datenobjekt vom Typ INT belegt im Speicher 2 Bytes. Mögliche INT – Werte sind die ganzen Zahlen von – 32768 bis + 32767 einschließlich.

REAL – Objekte

Jedes Datenobjekt vom Typ REAL belegt im Speicher 8 Bytes.

REALs haben eine 13 – stellige Mantisse, die im Rechner dezimal geführt wird. (Das heißt, bei Konversionen zwischen interner und TEXT – Darstellung treten keine Rundungsfehler auf.) Der Wertebereich wird durch folgende Eckwerte abgelegt:

9.999999999999e + 126	größter REAL – Wert
0.000000000001	kleinster positiver REAL – Wert mit x + 1.0 > 1.0
9.999999999999e – 126	kleinster positiver REAL – Wert > 0.0
– 9.999999999999e – 126	größter negativer REAL – Wert
– 9.999999999999e + 126	kleinster REAL – Wert

BOOL – Objekte

Jedes Datenobjekt vom Typ BOOL belegt im Speicher 2 Bytes.

TEXT – Objekte

Jedes Datenobjekt vom Typ TEXT besteht aus einem festen Teil von 16 Bytes und möglicherweise aus einem flexiblen Teil auf dem ***Heap***. Im festen Teil werden Texte bis zur Länge von 13 Zeichen untergebracht. Wenn eine TEXT – Variable einen Wert mit mehr als 13 Zeichen Länge annimmt, werden alle Zeichen auf dem Heap untergebracht. Genauer ergibt sich folgendes Bild:

kurzer Text (LENGTH < = 13):

Heap – Link	2 Bytes
Textlänge	1 Byte
Text	13 Bytes

langer Text (LENGTH > 13):

Heap – Link	2 Bytes
255	1 Byte
Länge	2 Bytes
ungenutzt	11 Bytes

Wenn eine Variable einmal Platz auf dem Heap bekommen hat, behält sie diesen vorbeugend auch dann, wenn sie wieder einen kurzen Text als Wert erhält. So muß wahrscheinlich kein neuer Platz auf dem Heap zugewiesen werden, wenn sie wieder länger wird. Das gilt allerdings nur bis zur nächsten Garbage Collection auf den TEXT – Heap, denn dabei werden alle Heap – Container minimal gemacht bzw. gelöscht, wenn sie nicht mehr benötigt werden. Der Platz auf dem Heap wird in Vielfachen von 16 Bytes vergeben. In Fremddatenräumen wird in jedem Container neben dem eigentlichen Text auch die Containerlänge untergebracht.

Beispiele:

TEXT – Länge	Speicherbedarf (Byte)
0	16
13	16
14	32
15	48
30	48
31	64
46	64
47	80
62	80

Die Heapgröße eines Fremddatenraums berechnet sich als:

1024 * 1024 – 520 = 1048056 – stat Bytes

'stat' ist dabei die statistische Größe der Datenstruktur, die dem Datenraum aufgeprägt wurde. Bei einem BOUND ROW 1000 TEXT ergibt sich also eine Heapgröße von

1048056 – (1000 * 16) = 1032056 Bytes.

ROW – und STRUCT – Objekte

Bei der Berechnung des Speicherbedarfs von STRUCTs und ROWs muß man bedenken, daß längere Datenobjekte ausgerichtet werden. Und zwar werden alle Objekte, die mindestens die Länge eines REAL – Objektes haben, auf durch 8 teilbare Speicheradressen ausgerichtet. Man bedenke, daß bei ROWs alle Elemente entsprechend ihres Elementtyps ausgerichtet sind.

Beispiele:	Länge (Byte)
ROW 2 BOOL	4
ROW 4 INT	8
ROW 5 INT	16
ROW 2 STRUCT (INT, BOOL)	4
ROW 100 STRUCT (INT,INT)	400
ROW 100 STRUCT (INT,REAL)	1600
ROW 100 STRUCT (INT,INT,INT,INT,REAL)	1600
ROW 100 STRUCT (REAL, REAL)	1600
ROW 100 STRUCT (INT,TEXT)	2400
ROW 100 STRUCT (INT,INT,INT,INT,TEXT)	2400
ROW 100 STRUCT (INT,TEXT,INT,TEXT)	4800
ROW 100 STRUCT (INT,INT,TEXT,TEXT)	4000
ROW 100 ROW 3 INT	600
ROW 100 ROW 4 INT	800
ROW 100 ROW 5 INT	1600
aber:	
ROW 500 INT	1000

Anmerkung: Bei der Speichervergabe der einfachen Variablen und Konstanten eines Programms spielen Verluste aufgrund von Ausrichtungen in der Regel keine Rolle. Der ELAN – Compiler optimiert dabei soweit möglich.

Teil 4: Standardpakete für Systemprogrammierer

1. Fehlerbehandlung

Übersicht

Fehler treten auf, wenn ein Programm eine gewünschte Leistung nicht erbringen kann. Solche Situationen müssen von System – Programmen kontrolliert behandelt werden. Die folgenden Ausführungen sind somit nur für diejenigen interessant, die "System" – Programme schreiben wollen.

Fehler treten in Operationen auf, wenn diese eine geforderte Leistung nicht erbringen können (z.B. Drucken einer nicht vorhandenen Datei). Da folgende Anweisungen aber davon ausgehen, daß die gewünschten Leistungen erbracht wurden, ist es nicht sinnvoll, die Operation weiter auszuführen. Wir sprechen vom Abbruch einer Operation, wenn nach einem Fehler keine Anweisungen mehr ausgeführt werden, sondern die Operation verlassen wird. Im EUMEL – System kann durch folgende drei Maßnahmen ein Abbruch verursacht werden:

- Aufruf der Prozedur 'errorstop':
  Die Operation wird mit einer Fehlermeldung abgebrochen, die man dem Aufruf von 'errorstop' als Parameter beifügt.

- Aufruf der Prozedur 'stop':
  Die Operation wird abgebrochen. Wirkt wie 'errorstop' mit der Meldung "stop".

- Umschalten in den Supervisor:
  Durch Betätigen der Taste SV und Eingabe des Kommandos 'halt'. Die laufende Operation wird abgebrochen. Wirkt wie ein 'errorstop', der von "außen" in das Programm induziert wird.

Da alle drei Maßnahmen zum Abbruch führen können und somit eine anomale (vorzeitige) Beendigung eines Programms bewirken, werden sie im folgenden zusammenfassend als Fehler bezeichnet.

Für solche Fehler bietet das EUMEL – System die Möglichkeit, den Abbruch zu unterdrücken. Dies kann notwendig werden, wenn

a) bestimmte Fehlerfälle vom aufrufenden Programm selbst behandelt werden sollen. Beispiel:

   Der EUMEL – Editor wird aufgerufen, um eine Datei zu editieren. Er versucht als erstes, die Datei zu assoziieren. Existiert die Datei nicht, wird die Prozedur (z.B. 'old'), mit der die Datei angemeldet werden soll, normalerweise mit der Fehlermeldung ' "datei" gibt es nicht' abgebrochen. Diesen Fehlerzustand fängt der Editor jedoch ab und versucht, eine neue Datei einzurichten. (Anmerkung: In Wirklichkeit fragt der Editor natürlich vor der Assoziierung mit 'exists' ab, ob die Datei existiert).

b) eine Operation die Kontrolle auf jeden Fall behalten soll.

   Dies ist z.B. beim Monitor notwendig. Gleich welche Fehler vom Monitor gerufene Programme produzieren, der Monitor muß in der Lage sein, die weitere Bearbeitung zu ermöglichen.

c) eine Operation nicht unterbrechbar sein darf.

   Beispielsweise dürfen Programm(teil)e, die Daten transportieren, nicht unterbrochen werden, da sonst ein Verlust dieser Daten eintreten könnte.

Fehlerbehandlung und Fängerebenen

Der Aufruf einer der Prozeduren

```
errorstop
stop
halt
```

(wobei letztere vom Supervisor gegeben werden muß) werden zusammenfassend als Fehler bezeichnet. Bei einem Fehler wird ein Fehlerzustand gesetzt. Im Fehlerzustand merkt sich das EUMEL – System, daß ein Fehler vorliegt. Die Prozeduren

```
enable stop
disable stop
```

bestimmen, ob Operationen im Fehlerzustand weiter bearbeitet oder abgebrochen werden. Beispiel:

```
INT VAR x;
get (x);
...
disable stop;
x := x * x;
...
```

Hier wird mit 'disable stop' verhindert, daß ein Abbruch beispielsweise durch 'INT – Ueberlauf' auftreten kann. Die Anweisungen nach 'x * x' werden also weiter bearbeitet.

Welchen Wert hat aber nun die Variable 'x', nachdem der Fehler auftrat? Offensichtlich war die den Fehler auslösende Operation '*' nicht in der Lage, den richtigen Wert zu errechnen. Abgebrochene Operationen liefern in der Regel keinen Wert. Dadurch ist der Wert von 'x' in unserem Beispiel nach einem Fehler bei '*' undefiniert. Es ist nun ersichtlich, daß mit der Anwendung der 'disable stop' – Prozedur äußerst vorsichtig zu verfahren ist, weil u.U. Werte verloren gehen können bzw. mit unerwarteten Werten weitergerechnet wird.

Damit Programmierer erfahren können, ob ein Fehler aufgetreten ist, gibt es die Informations – Prozedur

is error

über den Fehlerzustand. Die Prozedur liefert den Wert TRUE, wenn ein Fehler vorliegt, andernfalls FALSE. Die Prozedur

clear error

"löscht" den Fehlerzustand, d.h. anschließende Abfragen mit 'is error' liefern FALSE. (Die "richtige" Reaktion auf den Fehler muß ein Programmierer natürlich selbst bestimmen.) Beispiel:

```
INT VAR x;
get (x);
...
disable stop;
x := x * x;
IF is error
  THEN put ("'x'-Wert zu groß");
       x := 0;
       clear error
FI;
...
```

Leider würden jetzt aber auch alle folgenden Anweisungen bei eventuellen Fehlern nicht abgebrochen, also auch in Situationen, in denen ein Abbruch erwünscht ist, um Programmierfehler zu erkennen. Deshalb können durch

enable stop

Abbrüche wieder zugelassen werden. Wenn wir jetzt also schreiben:

```
INT VAR x;
get (x);
...
disable stop;
x := x * x;
IF is error
  THEN put ("'x'-wert zu gross");
       x := 0;
       clear error
FI;
enable stop;
...
```

dann würden – wie gewünscht – eventuelle Fehler in den Anweisungen nach 'enable stop' zu einem Abbruch führen.

Nicht mit 'clear error' gelöschte Fehler führen bei 'enable stop' ebenfalls zu einem Abbruch. In dem Programmteil

```
...
disable stop;
x := x * x;
enable stop;
...
```

würde der eventuell auftretender Fehler 'INT overflow' nicht abgefangen, sondern nur verzögert wirksam, weil er nicht mit 'clear error' gelöscht wurde.

Für die Behandlung von Fehlern durch Benutzer gibt es Prozeduren, die eine adäquate Reaktion auf den Fehler erlauben. Mit

error message

können Sie auf die erste Fehlermeldung (eines 'error stop') nach dem letzen 'clear error' zugreifen (d.h. Folgefehler verändern nicht die Originalmeldung). Die Prozedur

error code

liefert den Fehlercode, der bei der Prozedur 'errorstop' zusätzlich zum Fehlertext angegeben werden kann (zu der Festlegung von Fehlercodes vergl. S. 50).

error line

liefert die Zeilennummer des zuletzt aufgetretenen Fehlers. Mit

put error

kann eine noch anstehende Fehlermeldung ausgegeben werden. Beispiel:

```
INT VAR x;
get (x);
...
disable stop;
x := x * x;
IF is error
  THEN IF error message = "INT-Ueberlauf"
         THEN put ("'x'-wert zu gross");
         ELSE put error
       FI;
       clear error
FI;
enable stop;
...
```

Tritt ein Fehler auf, so wird die den Fehler auslösende Operation entweder abgebrochen oder "normal" weiter bearbeitet, je nachdem, ob 'enable stop' oder 'disable stop' gesetzt ist. Auf jeden Fall wird der Fehlerzustand an die aufrufende Operation weitergemeldet, die wiederum abgebrochen oder weiterbearbeitet werden kann usw. Die Weitermeldung eines Fehlers kann auch über mehrere Stufen erfolgen, solange bis der Fehler gelöscht wird. Andererseits gilt 'enable/ disable stop' nicht nur für die aktuelle Operation, sondern auch für gerufene Operationen ("Vererbung"). Die gerufenen Operationen können allerdings 'enable/disable stop' neu festlegen. Beispiel:

```
PROC a:                PROC b:          PROC c:
...                    ...              ROW 10 INT VAR x;
disable stop;          enable stop;     ...
b;                     ...              INT VAR i :: 4711;
IF is error            c;               x [i] := ...;
  THEN ...             ...              ...
       clear error     END PROC b       END PROC c
FI;
enable stop
END PROC a;
```

In der Prozedur 'a' wird die Prozedur 'b' aufgerufen. Diese ruft wiederum eine Prozedur 'c' auf. Für die Prozedur 'c' gilt nun der Zustand 'enable stop' der Prozedur 'b' (Vererbung von 'enable stop'). Tritt jetzt in 'c' der Subskriptions – Fehler auf, wird 'c' abgebrochen. Die Wirkung der fehlerauslösenden Operation ist nicht definiert.

Da aber auch die Prozedur 'b' im 'enable stop' Zustand ist, wird auch die Prozedur 'b' abgebrochen. Der Fehler bleibt jedoch erhalten, wird also weitergemeldet. Dies wirkt sich so aus, daß die Anweisung 'c' nicht ausgeführt wird. Da die Prozedur 'a' 'disable stop' gesetzt hat, werden die auf den Aufruf von 'b' folgenden Anweisungen durchlaufen und somit durch 'clear error' der Fehler gelöscht. In diesem Beispiel "fängt" die Prozedur 'a' Fehler auf, die in den Prozeduren 'b' und 'c' entstehen können.

Ein solcher Fänger wird durch zwei Prozeduren konstruiert. Der eigentliche Fänger (hier: Prozedur 'a') ruft eine ausführende Prozedur (hier: 'b') im 'disable stop' – Zustand auf. Die gerufene Prozedur setzt sofort 'enable stop' und führt dann die eigentlichen Aktionen aus. So wird die gerufene Prozedur abgebrochen (kann also im Fehlerfall nicht zuviel Schaden anrichten). Der Abbruch führt bis zur Fängerprozedur ('a') hinter den Aufruf der gerufenen Prozedur ('b'). Nach Löschung eventuell auftretender Fehler ist somit sichergestellt, daß der Fänger immer weiterarbeiten kann.

Wichtiger Hinweis

1. ***Da im 'disable stop' – Zustand kein Fehler zum Abbruch führt, kann eine Operation in diesem Zustand auch nicht durch 'halt' abgebrochen werden. Einerseits ist das für manche Systemteile wünschenswert, andererseits können Operationen, die auf Grund von Programmierfehlern nicht terminieren (Endlosschleifen), nicht unter Kontrolle gebracht werden. Also Vorsicht! (Letztes Mittel: Task löschen)***

2. ***Es ist nicht (!) garantiert, daß im Fehlerzustand aufgerufene Prozeduren ihre normale Wirkung haben. Garantiert ist dies jedoch für alle Prozeduren und Operatoren, die in diesem Kapitel aufgeführt werden.***

Merke: *Fehler sind im EUMEL – System Aufrufe der Prozeduren 'errorstop', 'stop' oder das Betätigen der SV Taste und dem Supervisor – Kommando 'halt'. Ein Fehler gilt solange, bis er mit Hilfe der Prozedur 'clear error' gelöscht wurde. Die Prozeduren 'enable/disable stop' steuern die Abarbeitung der Operationen im Fehlerfall. Gilt für eine Operation 'enable stop', wird die Operation abgebrochen, d.h. die restlichen Anweisungen der Operation nach der Fehler auslösenden Anweisung werden nicht durchlaufen. Ist 'disable stop' gesetzt, werden die restlichen Operationen weiterhin abgearbeitet. 'enable/disable stop' gilt für alle – auch indirekt – aufgerufenen Operationen ("Vererbung"), es sei denn, in den gerufenen Operationen wird ein erneutes 'enable/disable stop' gesetzt. Über die Aufrufkette werden ggf. auch die Fehler zurück gemeldet.*

Eine Fänger – Ebene ist eine Prozedur, die 'disable stop' setzt und dann andere Operationen aufruft. Nach jedem dieser Aufrufe kann eine Fehlerbehandlung mit 'clear error' durchgeführt werden. Damit ist gewährleistet, daß Fehler immer von der Fänger – Ebene "aufgefangen" und entsprechend behandelt werden.

Prozeduren zur Fehlerbehandlung

clear error

PROC clear error

Zweck: Löscht den Fehlerzustand. 'is error' liefert anschließend wieder FALSE. 'error message', 'error code' und 'error line' werden nicht gelöscht.

disable stop

PROC disable stop

Zweck: Unterbindet den Abbruch in aufgerufenen Operationen. 'disable stop' gilt für die Prozedur, in der sie aufgerufen wird und in allen folgenden gerufenen Prozeduren, es sei denn, sie wird durch 'enable stop' außer Kraft gesetzt. Wird die Operation verlassen, in der 'disable stop' aufgerufen wurde, wird der "alte" Zustand wiederhergestellt, der vor dem Aufruf der Operation galt. 'disable stop' kann weiterhin in einer aufgerufenen Operation durch den Aufruf von 'enable stop' in dieser und den folgenden Operationen außer Kraft gesetzt werden.

enable stop

PROC enable stop

Zweck: Setzt die Wirkung eines Aufrufs von 'disable stop' zurück. Fehler ('errorstop', 'stop' oder 'halt') in der aktuellen Operation oder den folgenden aufgerufenen Operationen führen zum Abbruch. Bisher nicht gelöschte Fehler (siehe 'clear error') führen sofort zum Abbruch.

error code

INT PROC error code

Zweck: Liefert den durch 'errorstop' gesetzten Fehlercode. Beispiel:

```
PROC test:
  enable stop;
  error stop (110, "Dies ist mein Abbruch!");
END PROC test;

...
disable stop;
test;
put (error code);   (* liefert 110 *)
clear error;
enable stop
```

error line

INT PROC error line

Zweck: Liefert die Zeilennummmer des Fehlers. (Voraussetzung : Die Übersetzung erfolgt im 'check on' – Modus).

error message

TEXT PROC error message

Zweck: Liefert die Fehlermeldung als Text. Anhand dieser Meldung kann entschieden werden, welcher Fehler vorliegt.

Hinweis: Eine Fehlermeldung "" (also: 'error stop ("")') führt zum Fehlerabbruch mit der Bedeutung "Fehlermeldung wurde bereits ausgegeben". Dementsprechend erfolgt bei der Fehlermeldung 'niltext' keine Reaktion bei 'put error'.

errorstop

PROC error stop (TEXT CONST message)

Zweck: Bricht ab und setzt die Zeilennummer (wenn man sich im 'checkon' – Modus befindet), in der der Fehler aufgetreten ist, sowie den Text 'message'. Der Abbruch kann mit 'disable stop' unterbunden werden. 'errorstop' hat keine Wirkung, wenn ein noch nicht gelöschter Fehler vorliegt. Zu einer Fehlermeldung "" siehe auch die Prozedur 'error message'. Als 'error code' wird 0 gesetzt.

PROC error stop (INT CONST code, TEXT CONST message)

Zweck: Analog obiger 'errorstop' – Prozedur, aber mit Angabe des Fehlercodes, der durch die Prozedur 'error code' in einer Fängerebene erfragt werden kann.

is error

BOOL PROC is error

Zweck: Informationsprozedur auf das Vorhandensein eines Fehlers.

put error

PROC put error

Zweck: Gibt die durch 'errorstop' gesetzte Fehlermeldung aus, falls ein Fehler noch nicht gelöscht ist (siehe auch: 'error message').

Fehlercodes

Einige Fehlercodes sind bereits belegt:

0	kein Fehlercode spezifiziert (Standardwert)
1	'halt' vom Terminal
2	Stack – Ueberlauf
3	Heap – Ueberlauf
4	INT – Ueberlauf
5	DIV durch 0
6	REAL – Ueberlauf
7	TEXT – Ueberlauf
8	zu viele DATASPACEs
9	Ueberlauf bei Subskription
10	Unterlauf bei Subskription
11	falscher DATASPACE – Zugriff
12	INT nicht initialisiert
13	REAL nicht initialisiert
14	TEXT nicht initialisiert
15	nicht implementiert
16	Block unlesbar
17	Codefehler
100	Syntax – Fehler beim Übersetzen

2. THESAURUS

Ein Thesaurus ist ein Namensverzeichnis, das bis zu 200 Namen beinhalten kann. Dabei muß jeder Namen mindestens ein Zeichen und darf höchstens 100 Zeichen lang sein. Steuerzeichen (code < 32) sind in Namen nicht erlaubt.

Ein Thesaurus ordnet jedem eingetragenen Namen einen Index zwischen 1 und 200 (einschließlich) zu. Diese Indizes bieten dem Anwender die Möglichkeit, Thesauri zur Verwaltung benannter Objekte zu verwenden. (Der Zugriff erfolgt dann über den Index eines Namens in einem Thesaurus). So werden Thesauri u.a. von der Dateiverwaltung benutzt. Sie bilden die Grundlage der ALL – und SOME – Operatoren.

Grundoperationen

CONTAINS

BOOL OP CONTAINS (THESAURUS CONST t, TEXT CONST name)

Zweck: Liefert genau dann TRUE, wenn 't' den Namen 'name' enthält. Falls 'name = ""' oder 'LENGTH name > 100', wird FALSE geliefert.

delete

PROC delete (THESAURUS VAR t, TEXT CONST name, INT VAR index)

Zweck: Falls der Name 'name' im Thesaurus 't' enthalten ist, wird er dort gelöscht. In 'index' wird dann sein alter Index geliefert, unter dem er im Thesaurus eingetragen war. Ist der Name nicht im Thesaurus enthalten, wird 0 als Index geliefert.

PROC delete (THESAURUS VAR t, INT CONST index)

Zweck: Der Eintrag mit dem angegebenen Index wird aus dem Thesaurus 't' gelöscht.

empty thesaurus

THESAURUS PROC empty thesaurus

Zweck: Für Initialisierungszwecke wird ein leerer Thesaurus geliefert.

get

PROC get (THESAURUS CONST t, TEXT VAR name, INT VAR index)

Zweck: Liefert den "nächsten" Eintrag aus dem Thesaurus 't'. "Nächster" heißt hier, der kleinste vorhandene mit einem Index größer als 'index'. Dabei wird in 'name' der Name und in 'index' der Index des Eintrags geliefert. D.h. 'index' wird automatisch weitergeschaltet. Den ersten Eintrag erhält man entsprechend durch Aufruf mit 'index = 0'. Nach dem letzten Eintrag wird 'name = ""' und 'index = 0' geliefert. Beispiel:

```
TEXT VAR name;
INT VAR index := 0 ;
get (thesaurus, name, index) ;
WHILE index > 0 REP
  putline (name) ;
  get (thesaurus, name, index)
PER
```

highest entry

INT PROC highest entry (THESAURUS CONST t)

Zweck: Liefert den höchsten belegten Index des Thesaurus 't'.

Achtung: Das ist nicht die Anzahl der vorhandenen Namen, da durch Löschungen Lücken entstanden sein können.

insert

PROC insert (THESAURUS VAR t, TEXT CONST name, INT VAR index)

Zweck: Der Name 'name' wird als zusätzlicher Eintrag in den Thesaurus 't' eingetragen und der dafür vergebene Index geliefert. Falls der Thesaurus schon voll ist und der Name nicht mehr eingetragen werden kann, wird 0 als Index geliefert.

Achtung: Mehrfacheintragungen sind möglich. Wenn man diese verhindern will, muß man entsprechend vermittels

```
IF NOT t CONTAINS name
  THEN insert (t, name, index)
FI
```

eintragen.

Fehlerfall:

* Name unzulaessig

PROC insert (THESAURUS VAR t, TEXT CONST name)
Zweck: s.o. Allerdings wird der Index des Namens nicht geliefert. Ein Thesaurusüberlauf wird entsprechend als 'errorstop' gemeldet.
Fehlerfälle:
* Name unzulaessig
* THESAURUS – Ueberlauf

link

INT PROC link (THESAURUS CONST t, TEXT CONST name)
Zweck: Liefert den Index des Namens 'name' im Thesaurus 't'. Falls der Name nicht enthalten ist, wird 0 geliefert. Ist der Name mehrfach im Thesaurus enthalten, ist nicht definiert, welcher der möglichen Indizes geliefert wird.

name

TEXT PROC name (THESAURUS CONST t, INT CONST index)
Zweck: Liefert den Namen des Eintrags mit dem Index 'index' aus dem Thesaurus 't'. Falls kein solcher Eintrag im Thesaurus enthalten ist, wird Niltext geliefert.

rename

PROC rename (THESAURUS VAR t, TEXT CONST old, new)
Zweck: Ändert im Thesaurus 't' einen Eintrag mit dem alten Namen 'old' in 'new' um. Falls 'old' nicht im Thesaurus enthalten ist, wird keine Leistung erbracht. Falls 'old' mehrfach in 't' enthalten ist, ist nicht definiert, welcher der möglichen Einträge geändert wird.
Fehlerfall:
* Name unzulaessig

PROC rename (THESAURUS VAR t, INT CONST index, TEXT CONST new)
Zweck: Ändert im Thesaurus 't' den Namen des durch 'index' identifizierten Eintrags in 'new'.
Fehlerfall:
* Name unzulaessig

THESAURUS

TYPE THESAURUS
Zweck: Bezeichnet Thesaurus – Datenobjekte

:=

OP := (THESAURUS VAR dest, THESAURUS CONST source)
Zweck: Zuweisung

Verknüpfungsoperationen

Das Paket 'nameset' bietet die Möglichkeit, Operationen nicht nur auf einzelnen Dateien, sondern auf (geordneten) Mengen ablaufen zu lassen:

ALL

THESAURUS OP ALL (TASK CONST task)
Zweck: Liefert einen Thesaurus, der alle Dateinamen der angegebenen Task enthält.

THESAURUS OP ALL (TEXT CONST file name)
Zweck: Liefert einen Thesaurus, der die in der angegebenen Datei vorhandenen Namen (jede Zeile ein Name) enthält.

all

THESAURUS PROC all
Zweck: Liefert einen Thesaurus, der alle Dateinamen der eigenen Task enthält. Entspricht 'ALL myself'.

LIKE

THESAURUS OP LIKE (THESAURUS CONST thesaurus, TEXT CONST muster)
Zweck: Alle im Thesaurus enthaltenen Dateien, die dem 'muster' entsprechen sind im Ergebnisthesaurus enthalten.
(Die Syntax von 'muster' findet man bei der Beschreibung des Pattern – Matching)

SOME

THESAURUS OP SOME (THESAURUS CONST thesaurus)
Zweck: Bietet den angegebenen Thesaurus zum editieren an. Dort können nicht erwünschte Namen gestrichen werden.

THESAURUS OP SOME (TASK CONST task)
Zweck: Aufruf von: SOME ALL task.

THESAURUS OP SOME (TEXT CONST file name)
Zweck: Aufruf von: SOME ALL filename.

FILLBY

OP FILLBY (THESAURUS VAR thesaurus, FILE VAR file)
Zweck: Schreibt 'file' in den Thesaurus. Dabei werden Zeilen, die schon im Thesaurus sind, nicht mehr in den Thesaurus geschrieben. Jede Zeile kommt im Thesaurus also nur einmal vor.

OP FILLBY (FILE VAR file, THESAURUS CONST thesaurus)
Zweck: Schreibt den Thesaurus in die Datei 'file'.

OP FILLBY (TEXT CONST filename,
THESAURUS CONST thesaurus)
Zweck: Richtet eine Datei mit dem Namen 'filename' ein und schreibt den Thesaurus in die Datei.

+

THESAURUS OP + (THESAURUS CONST left, right)
Zweck: Liefert die Vereinigungsmenge von 'left' und 'right'.
Achtung: Die Vereinigungsmenge enthält keine Namen mehrfach.

THESAURUS OP + (THESAURUS CONST left, TEXT CONST right)
Zweck: Fügt dem Thesaurus 'right' zu, wenn 'right' noch nicht im Thesaurus enthalten ist.

–

THESAURUS OP – (THESAURUS CONST left, right)
Zweck: Liefert die Differenzmenge. Achtung: Die Differenzmenge enthält keine Namen mehrfach.

THESAURUS OP – (THESAURUS CONST left, TEXT CONST right)
Zweck: Nimmt den Namen 'right' aus dem Thesaurus.

/

THESAURUS OP / (THESAURUS CONST left, right)
Zweck: Liefert die Schnittmenge
Achtung: Die Schnittmenge enthält keine Namen mehrfach.

do

PROC do (PROC (TEXT CONST) operate, THESAURUS CONST thesaurus)
Zweck: Ruft 'operate' nacheinander mit allen im Thesaurus enthaltenen Namen auf.

PROC do (PROC (TEXT CONST, TASK CONST) operate,
THESAURUS CONST thesaurus, TASK CONST task)
Zweck: s.o.

erase

PROC erase (THESAURUS CONST thesaurus)
Zweck: Löscht alle aufgeführten Dateien in der Vater – Task.

PROC erase (THESAURUS CONST thesaurus, TASK CONST manager)
Zweck: Löscht alle aufgeführten Dateien in der Task 'manager'.

fetch

PROC fetch (THESAURUS CONST thesaurus)
Zweck: Holt alle aufgeführten Dateien vom Vater.

PROC fetch (THESAURUS CONST thesaurus, TASK CONST manager)
Zweck: Holt alle aufgeführten Dateien vom 'manager'.

fetch all

PROC fetch all (TASK CONST manager)
Zweck: Holt alle Dateien vom 'manager'. Diese Prozedur entspricht dem Aufruf der Prozedur 'fetch (ALL manager, manager)'.

PROC fetch all
Zweck: Aufruf der Prozedur 'fetch all (father)'.

forget

PROC forget (THESAURUS CONST thesaurus)
Zweck: Löscht alle aufgeführten Dateien in der Benutzer – Task.

insert

PROC insert (THESAURUS CONST thesaurus)
Zweck: Insertiert alle aufgeführten Dateien in der Benutzer – Task.

remainder

PROC remainder

Zweck: Liefert nach einem 'errorstop' die noch nicht bearbeiteten Dateien.

Beispiel:

'save all (archive)'

kann dazu führen, daß nicht alle Dateien auf das Archiv geschrieben werden können. Fehlermeldung:

'"....." kann nicht geschrieben werden (Archiv voll)'

Nachdem man eine neue Floppy ins Archivlaufwerk gelegt hat, kann man mit

'save (remainder, archive)'

den Rest der Dateien auf der Floppy sichern.

save

PROC save (THESAURUS CONST thesaurus)

Zweck: Schickt alle aufgeführten Dateien zur Vater – Task.

PROC save (THESAURUS CONST thesaurus, TASK CONST manager)

Zweck: s.o.

save all

PROC save all (TASK CONST manager)

Zweck: Schickt alle eigenen Dateien zum 'manager'. Diese Prozedur entspricht dem Aufruf der Prozedur 'save (ALL myself, manager)'.

PROC save all

Zweck: Aufruf der Prozedur 'save all (father)'.

Beispiele:

```
save (ALL myself)
forget (ALL myself)
forget (all)
fetch (SOME father)
fetch (ALL father - ALL myself)
insert (ALL "gen datei")
fetch (ALL myself - ALL archive, archive)
```

3. Kommandos und Dialog

Kommandodialog

Das Paket "command dialogue" dient zur zentralen Steuerung und einfachen Durchführung von Kommando – Dialogen wie

"datei" loeschen (j/n)?

Er wird von allen Systemteilen verwandt, die einen Kommandodialog mit dem Benutzer aufnehmen. Anwenderprozeduren mit ähnlichen Problemen sollten genauso damit arbeiten.

Der Kommandodialog kann zentral aus – und eingeschaltet werden.

command dialogue

BOOL PROC command dialogue

Zweck: Liefert den aktuellen Zustand des Kommandodialogs:

TRUE – Dialog soll geführt werden!

FALSE – Dialog soll nicht geführt werden!

PROC command dialogue (BOOL CONST status)

Zweck: Schaltet den Kommandodialog ein ('status' = TRUE) oder aus ('status' = FALSE). Der alte Zustand wird überschrieben. Soll später wieder in den alten Zustand zurückgeschaltet werden, muß er vorher erfragt und gesichert werden.

yes

BOOL PROC yes (TEXT CONST question)

Zweck: a) command dialogue – – > TRUE

Der übergebene Fragetext wird durch " (j/n)?" ergänzt auf dem Terminal ausgegeben. Als Antwort wird eine der Tasten <j>, <J>, <y>, <Y>, <n>, <N> akzeptiert; jede andere Eingabe führt zu einem akustischen Signal und der Fragewiederholung. Das Resultat der Prozedur ist

TRUE bei bejahender Antwort (j,J,y,Y)

FALSE bei verneinender Antwort (n,N)

b) command dialogue – – > FALSE

Keine Aktion, das Resultat ist TRUE.

no

BOOL PROC no (TEXT CONST question)

Zweck: a) command dialogue – – > TRUE

Frage und Antwort wie bei 'yes'. Das Resultat ist

TRUE bei verneinender Antwort (n,N)

FALSE bei bejahender Antwort (j,J,y,Y)

b) command dialogue – – > FALSE

Keine Aktion, das Resultat ist FALSE.

say

PROC say (TEXT CONST message)

Zweck: IF command dialogue THEN out (text) FI

last param

TEXT PROC last param

Zweck: Liefert den zuletzt gesetzten Parameter – Text (siehe folgende Prozedur). Falls 'command dialogue' = TRUE und die 'param position' > 0 ist, wird der Parametertext als Standardparameter an der angegebenen x – Position eine Zeile höher in der Form ("...") ausgegeben. Diese Prozedur wird von den parameterlosen Kommandos bzw. Prozeduren wie 'edit', 'run' usw. verwandt, um mit dem Standardparameter weiterzuarbeiten.

PROC last param (TEXT CONST new)

Zweck: Setzt 'last param' auf 'new'. (Das Setzen muß explizit durchgeführt werden und geschieht nicht implizit durch den 'command handler' (s.dort)!) 'Last param' wird beispielsweise von den einparametrigen Prozeduren 'edit' und 'run' gesetzt.

param position

PROC param position (INT CONST x)

Zweck: Setzt die Echoposition für 'last param'. Bei x = 0 wird ein Echo unterdrückt.

std

TEXT PROC std

Zweck: Liefert wie 'last param' den zuletzt gesetzten Parameter. Im Gegensatz dazu wird der Parameter aber nicht ausgegeben.

Kommandoverarbeitung

Das Paket 'command handler' stellt Prozeduren zur Kommandoanalyse und zum Führen des kompletten Kommandodialogs zur Verfügung.

get command

PROC get command (TEXT CONST dialogue text, TEXT VAR command line)

Zweck: Falls eine Fehlermeldung aussteht, ('is error' liefert TRUE), wird sie über 'put error' ausgegeben und der Fehlerzustand zurückgesetzt. Der 'dialogue text' wird als Dialogaufforderung ausgegeben und der Benutzer kann eine Kommandozeile eingeben. Die letzte Kommandozeile wird ihm dabei automatisch (zum Editieren) angeboten, wenn vorher eine Fehlermeldung anstand. Der Benutzer kann das auch sonst erreichen, wenn er zu Beginn <ESC k> gibt. Die Kommandozeile wird dem Aufrufer in der Variablen 'command line' geliefert.

PROC get command (TEXT CONST dialogue text)

Zweck: s.o. Allerdings wird eine interne Kommandozeile des Pakets 'command handler' als 'command line' verwandt. Dadurch wird es möglich, alle Spuren einer Kommandoeingabe durch 'cover tracks' (s. dort) zu beseitigen.

analyze command

PROC analyze command (TEXT CONST command list, command line,
INT CONST permitted type,
INT VAR command index, number of params,
TEXT VAR param 1, param 2)

Zweck: Die übergebene Kommandozeile ('command line') wird anhand der übergebenen 'command list' analysiert. Sie ist ein TEXT, der aus einer Folge von Kommandospezifikationen besteht. Jede hat die Form

K:I.P

- K Kommandotext, Prozedurname nach ELAN – Syntax
- I Hauptindex, Form eines INT – Denoters
- P Parameterspezifikation, eine Folge der Ziffern 0, 1 und 2.

Beispiele:

- 'edit:15.012'
  Das Kommando 'edit' wird in drei verschieden parametrisierten Formen spezifiziert:
  edit mit 0 Parameter erhält Index 15
  edit mit 1 Parameter erhält Index 16
  edit mit 2 Parametern erhält Index 17

- 'fetch:18.1'
  Das Kommando 'fetch' wird in einer Form spezifiert:
  fetch mit 1 Parameter erhält Index 18

Die Analyse erfolgt gemäß ELAN – Syntaxregeln. Dabei sind als Parameter Denoter vom Typ TEXT und vom übergebenen ' permitted type' zugelassen. Diese Typen werden wie beim Scanner (s. Benutzerhandbuch Kap. 11.1) angegeben:

1 tag
2 bold
3 number
4 text
5 operator
6 delimiter

Falls das Kommando in der Kommandoliste gefunden wird (und die Syntax in Ordnung ist), wird der entsprechende 'command index' zurückgemeldet. Die Parameter werden (falls vorhanden) in 'param 1' und 'param 2' abgelegt. Undefinierte oder nicht vorhandene Parameter werden als Niltext geliefert. Wenn ein Kommando vorhanden ist, die Anzahl der Parameter aber nicht stimmt, wird der negative Hauptindex geliefert. Ist es vollkommen unbekannt oder ist die Eingabe zu komplex (mehrere Kommandos, Ausdrücke oder komplexere ELAN – Statements), wird 0 geliefert. Der Anwender kann in solchen Fällen die Analyse mit einer anderen Kommandoliste fortsetzen, das Kommando dem ELAN – Compiler übergeben oder eine Fehlermeldung auslösen (s. 'command error').

PROC analyze command (TEXT CONST command list,
INT CONST permitted type,
INT VAR command index, number of params,
TEXT VAR param 1, param 2)

Zweck: s.o. Allerdings wird die interne Kommandozeile des Pakets 'command handler' als 'command line' verwandt.

command error

PROC command error

Zweck: Falls bei der Kommandoanalyse ein Fehler gefunden wurde, führt er nicht zum 'errorstop', sondern wird nur hinterlegt. (Soll das Kommando dem Compiler übergeben werden, liegt ja evt. überhaupt kein Fehler vor.) Diese hinterlegte Meldung kann mit 'command error' als 'errorstop' gegeben werden. Mögliche Meldungen:

"ungueltiger name"
") fehlt"
"(fehlt"
"Parameter ist kein TEXT ("fehlt)"
"Kommando zu schwierig"

cover tracks

PROC cover tracks

Zweck: Die Spuren der letzten Kommandoanalyse werden gelöscht. Das dient u.a. dazu, daß später eingerichtete Sohntasks keine Relikte des Kommandos mehr auf dem Textheap vorfinden und evtl. vermittels nicht initialisierter TEXT VARs herausfinden können. Vollständig können die Spuren aber nur dann gelöscht werden, wenn für die Kommandoanalyse die 'get command' – und 'analyze command' – Prozeduren benutzt wurden, die auf der internen Kommandozeile des Pakets 'command handler' arbeiten.

do command

PROC do command

Zweck: Die interne Kommandozeile des Pakets 'command handler' wird dem ELAN – Compiler zur Ausführung übergeben.

Beispiele zur Kommandoverarbeitung

Kleiner Monitor

```
LET command list = "otto:1.12emil:3.012hugo:6.0" ;

LET   number = 3 ,
      text   = 4 ;

INT VAR command index, params ;
TEXT VAR param 1, param 2 ;

PROC monitor :

     disable stop ;
     command dialogue (TRUE) ;
     REP  get command ("gib kleines kommando:") ;
          analyze command (command list, text,
                           command index, params,
                           param 1, param 2) ;
          execute command
     PER

ENDPROC monitor ;

PROC execute command :

     enable stop ;
     SELECT command index OF
          CASE 1 : otto (param 1)
          CASE 2 : otto (param 1, param 2)
          CASE 3 : emil
          CASE 4 : emil (param 1)
          CASE 5 : emil (param 1, param 2)
          CASE 6 : hugo
       OTHERWISE do command line
     END SELECT

ENDPROC execute command ;
```

Steuerkommando – Analyse

```
PROC command (TEXT CONST command text) :

     disable stop ;
     command dialoge (FALSE) ;
     analyze command (command list, command text, number,
                      command index, params, param 1, param 2) ;
     execute command ;
     IF is error
       THEN put error ;
            clear error
     FI

ENDPROC command ;

PROC execute command :

     enable stop ;
     SELECT command index OF
          CASE ....
        OTHERWISE    IF command index = 0
                        THEN errorstop ("unbekanntes Kommando")
                        ELSE command error
                     FI
     END SELECT

ENDPROC execute command ;
```

4. Verschiedenes

SESSION

Mit Hilfe von 'session' kann man feststellen, ob das System neu gestartet wurde. Dabei spielt es keine Rolle, ob es korrekt ('shutup') abgeschaltet wurde, oder ob es sich um einen "Rerun" handelt.

session

INT PROC session

Zweck: Liefert eine "Sitzungsnummer". Diese wird automatisch bei jedem Systemstart erhöht.

Beispiel:

```
REP
  INT VAR old session := session ;
  WHILE session = old session REP pause (100) PER ;
  putline ("Neuer Systemstart")
PER.
```

INITFLAG

Im Multi – User – System ist es oft notwendig, Pakete beim Einrichten einer neuen Task in dieser neu zu initialisieren. Das muß z.B. bei der Dateiverwaltung gemacht werden, da die neue Task ja nicht die Dateien des Vaters erbt. Mit Hilfe von INITFLAG – Objekten kann man zu diesem Zweck feststellen, ob ein Paket ***in dieser Task*** schon initialisiert wurde.

INITFLAG

TYPE INITFLAG

Zweck: Erlaubt die Deklaration entsprechender Flaggen.

:=

OP := (INITFLAG VAR flag, BOOL CONST flagtrue)

Zweck: Erlaubt die Initialisierung von INITFLAGs

initialized

BOOL PROC initialized (INITFLAG VAR flag)

Zweck: Wenn die Flagge in der Task A auf TRUE oder FALSE gesetzt wurde, dann liefert sie beim ersten Aufruf den entsprechenden Wert, danach immer TRUE (in der Task A!).
Beim Einrichten von Söhnen wird die Flagge in den Sohntasks automatisch auf FALSE gesetzt. So wird erreicht, daß diese Prozedur in den neu eingerichteten Söhnen und Enkeltasks genau beim ersten Aufruf FALSE liefert.

Beispiel:

```
PACKET stack DEFINES push, pop:

INITFLAG VAR in this task := FALSE ;
INT VAR stack pointer ;
ROW 1000 INT VAR stack ;

PROC push (INT CONST value) :

  initialize stack if necessary ;
  ....

ENDPROC push ;

PROC pop (INT VAR value) :

  initialize stack if necessary ;
  ....

ENDPROC pop ;.

initialize stack if necessary :
  IF NOT initialized (in this task)
    THEN stack pointer := 1
  FI .

ENDPACKET stack
```

Bit – Handling

Die Bit – Operationen arbeiten auf INT – Objekten. Sie können z.B. für die Systemprogrammierung benutzt werden, wenn es um Bitmasken u.ä. geht.

Ein INT besteht aus 16 Bits. Dabei hat das niederwertigste die Nummer 0, das höchstwertige die Nummer 15.

AND

INT OP AND (INT CONST left, right)
Zweck: Bitweise UND – Verknüpfung von 'left' mit 'right'.

OR

INT OP OR (INT CONST left, right)
Zweck: Bitweise ODER – Verknüpfung von 'left' mit 'right'.

XOR

INT OP XOR (INT CONST left, right)
Zweck: Bitweise EXCLUSIV – ODER – Verknüpfung von 'left' mit 'right'.

bit

BOOL PROC bit (INT CONST bits, bit no)
Zweck: Liefert TRUE genau dann, wenn das Bit mit der Nummer 'bit no' in dem INT 'bits' gesetzt ist.

set bit

PROC set bit (INT VAR bits, INT CONST bit no)
Zweck: Das Bit mit der Nummer 'bit no' wird in 'bits' auf 1 gesetzt.

reset bit

PROC reset bit (INT VAR bits, INT CONST bit no)
Zweck: Das Bit mit der Nummer 'bit no' wird in 'bits' auf 0 gesetzt.

rotate

PROC rotate (INT VAR bits, INT CONST number of bits)
Zweck: Bits können mit dieser Prozedur zyklisch geschiftet werden.

```
Bsp.: rotate (1,1)        - - - > 2
      rotate (1,2)        - - - > 4
      rotate (1, - 3)     - - - > 16384
      rotate (16384,3)    - - - > 1
```

lowest set

INT PROC lowest set (INT CONST bits)

Zweck: Liefert die Nummer des niederwertigsten 1 – Bits in 'bits'. Ist kein Bit auf 1 gesetzt, wird – 1 geliefert.

lowest reset

INT PROC lowest reset (INT CONST bits)

Zweck: Liefert die Nummer des niederwertigsten 0 – Bits in 'bits'. Ist kein Bit auf 0 gesetzt, wird – 1 geliefert.

5. Blockorientierte Ein – /Ausgabe

Die blockorientierte Ein – /Ausgabe dient dazu, Datenraumseiten (Blöcke) oder Teile davon über die Kanäle zu transferieren. Sie wird vom System u.a. beim Archivzugriff und bei der Konfigurierung der Kanäle eingesetzt.

Die Wirkung der blockorientierten Ein – /Ausgabeoperationen kann dabei kanal – und rechnerspezifisch unterschiedlich sein.
Auf dem Archivkanal (31) und allen anderen Block – IO – Kanälen werden bei 'code 1 = 0' die normalen Blocklese – bzw. – schreibeoperationen durchgeführt. 'code 2' gibt dabei die Blocknummer an. Andere (positive) Werte von 'code 1' sind zur Zeit nicht offiziell definiert. Negative Werte können von den SHards für Spezialaufgaben vergeben werden.

blockin

PROC blockin (DATASPACE VAR ds, INT CONST page nr, code1, code2,
INT VAR return code)

Zweck: Die Seite 'page nr' des Datenraums 'ds' wird "eingelesen". Die Operation kann durch 'code1' und 'code2' näher gesteuert werden.

PROC blockin (ROW 256 INT VAR block, INT CONST code1, code2,
INT VAR return code)

Zweck: Wie oben, nur wird der Block direkt als Datenstruktur übergeben.

blockout

PROC blockout (DATASPACE CONST ds, INT CONST page nr,
code1, code2, INT VAR return code)

Zweck: Die Seite 'page nr' des Datenraums 'ds' wird "ausgegeben". Die Operation kann durch 'code1' und 'code2' näher gesteuert werden.

PROC blockout (ROW 256 INT CONST block, INT CONST code1, code2,
INT VAR return code)

Zweck: Wie oben, nur wird der Block als Datenstruktur übergeben.

control

PROC control (INT CONST code1, code2, code3, INT VAR return code)

Zweck: Diese Prozedur dient zur Kanalsteuerung.

ds pages

INT PROC ds pages (DATASPACE CONST ds)

Zweck: Liefert die Anzahl der belegten Seiten eines Datenraums. (Jede Seite ist 512 Byte groß.)

next ds page

INT PROC next ds page (DATASPACE CONST ds, INT CONST page nr)

Zweck: Liefert die Nummer der nächsten (von 'page nr' an gerechneten) Seite des Datenraums. Die erste belegte Seite erhält man durch

```
next ds page (ds, -1)
```

Achtung: Die Seitennummern müssen nicht lückenlos sein.

Teil 5: Supervisor, Tasks und Systemsteuerung

1. Tasks

Der Datentyp TASK

Benannte Tasks werden innerhalb eines Rechners vollständig und eindeutig über ihren Namen identifiziert. Eine weitere Möglichkeit der Identifikation besteht in der Verwendung von Datenobjekten vom Typ TASK. Beispiel:

```
TASK VAR plotter := task ("PLOTTER 1")
```

Die Taskvariable 'plotter' bezeichnet jetzt die Task im System, die augenblicklich den Namen "PLOTTER 1" hat. Nun sind Taskvariablen auch unter Berücksichtigung der Zeit und nicht nur im aktuellen Systemzustand eindeutig. Der Programmierer braucht sich also keine Sorgen darüber zu machen, daß seine Taskvariable irgendwann einmal eine "falsche" Task (nach Löschen von "PLOTTER 1" neu eingerichtete gleichen oder anderen Namens) identifiziert. Wenn die Task "PLOTTER 1" gelöscht worden ist, bezeichnet 'plotter' keine gültige Task mehr.

Unbenannte Tasks haben alle den Pseudonamen " – ". Sie können nur über Taskvariablen angesprochen werden.

Der Task – Katalog wird vom Supervisor geführt; andere Tasks können sich Kopien dieses Katalogs besorgen. Einige Prozeduren arbeiten auf dieser taskeigenen Kopie, ohne diese automatisch auf den neuesten Stand zu bringen (Effizienzgründe). Das muß bei Bedarf explizit geschehen.

TASK

TYPE TASK
Zweck: Interner Taskbezeichner

:=

OP := (TASK VAR dest, TASK CONST source)
Zweck: Zuweisung von internen Taskbezeichnern

=

BOOL OP = (TASK CONST left, right)
Zweck: Gleichheitsabfrage

<

BOOL OP < (TASK CONST left, right)
Zweck: Überprüft, ob die Task 'left' ein Sohn, Enkel, Urenkel, ... der Task 'right' ist.

/

TASK OP / (TEXT CONST task name)
Zweck: Liefert die Task des angegebenen Namens, falls sie existiert. Der eigene Katalog wird automatisch aktualisiert (identisch mit der PROC task (TEXT CONST task name).
Fehlerfall:
* ... gibt es nicht

TASK OP / (INT CONST station number, TEXT CONST name)
Zweck: Liefert die Task des angegebenen Namen von der Station mit der angegebenen Nummer.

access

PROC access (TASK CONST task)
Zweck: Aktualisiert den eigenen Taskkatalog, falls 'task' nicht darin enthalten ist.

access catalogue

PROC access catalogue
Zweck: Aktualisiert den eigenen Taskkatalog, indem die neueste Fassung vom Supervisor geholt wird. Die Prozeduren 'father', 'son', 'brother' arbeiten dann auf dieser neuen Fassung.

archive

TASK PROC archive

Zweck: Liefert den internen Taskbezeichner der aktuellen Task mit Namen "ARCHIVE". Diese Prozedur dient zum schnellen und bequemen Ansprechen der Archivtask.

brother

TASK PROC brother (TASK CONST task)

Zweck: Liefert den nächsten Bruder von 'task'. Falls kein Bruder existiert, wird 'niltask' geliefert. Aktualisiert den eigenen Katalog nicht automatisch!

canal

TASK PROC canal (INT CONST channel number)

Zweck: Diese Prozedur zeigt an, welche Command – Analyser – Task an einem bestimmten Kanal hängt.

exists

BOOL PROC exists (TASK CONST task)

Zweck: Falls 'task' auf der eigenen Station liegt, informiert diese Prozedur, ob die angegebene 'task' noch existiert. Der eigene Taskkatalog wird dabei aktualisiert.

Wenn abgefragt werden soll, ob 'task' auf einer anderen Station liegt, muß die Prozedur 'name (task) <> "" ' verwendet werden.

Achtung: Diese Prozedur taugt nicht dazu, zu erfragen, ob eine Task mit bestimmtem Namen im System exisiert.

```
exists (task ("hugo"))
```

Fall die Task "hugo" nicht existiert, führt nämlich schon der Aufruf 'task ("hugo")' zum 'errorstop ("hugo" gibt es nicht)'.

exists task

BOOL PROC exists task (TEXT CONST name)

Zweck: Wenn auf der eigenen Station eine Task mit dem Namen 'name' existiert, liefert diese Prozedur 'TRUE'.

father

TASK PROC father

Zweck: Liefert die eigene Vatertask.

TASK PROC father (TASK CONST task)

Zweck: Liefert den Vater von 'task'. Existiert kein Vater (z.B. bei UR), wird niltask geliefert. Aktualisiert den eigenen Katalog nicht automatisch!

index

INT PROC index (TASK CONST task)

Zweck: Liefert einen INT – Wert von 1 bis 125, der 'task' unter allen gleichzeitig (!) existierenden Tasks eindeutig identifiziert.

is niltask

BOOL PROC is niltask (TASK CONST task)

Zweck: task = niltask

myself

TASK PROC myself

Zweck: Liefert eigenen Task – Bezeichner.

name

TEXT PROC name (TASK CONST task)

Zweck: Liefert den Namen von 'task'. Die Task muß noch im System existieren, sonst ist der Name nicht mehr bekannt. Falls die 'task' noch nicht im eigenen Katalog enthalten ist, wird er aktualisiert.

niltask

TASK CONST niltask

Zweck: Bezeichner für "keine Task". So liefern die Prozeduren 'son', 'brother' und 'father' als Resultat 'niltask', wenn keine Sohn –, Bruder – oder Vatertask existiert.

printer

TASK PROC printer

Zweck: Liefert den internen Taskbezeichner der aktuellen Task mit Namen PRINTER. Diese Prozedur dient zum schnellen und bequemen Ansprechen des Druckspoolers.

public

TASK PROC public

Zweck: Liefert den internen Taskbezeichner der Task PUBLIC.

reserve

PROC reserve (TASK CONST task)

Zweck: Reservieren einer Task für den ausschließlichen Dialog mit der Task, in der das Kommando gegeben wurde.

PROC reserve (TEXT CONST message, TASK CONST task)

Zweck: Wie 'reserve (TASK CONST task)' mit Übergabe einer 'message'.

son

TASK PROC son (TASK CONST task)

Zweck: Liefert den ersten Sohn von 'task'. Falls keiner im Katalog vermerkt ist, wird 'niltask' geliefert. Aktualisiert den eigenen Katalog nicht automatisch!

supervisor

TASK PROC supervisor

Zweck: Liefert den internen Taskbezeichner des Supervisors.

task

TASK PROC task (TEXT CONST task name)

Zweck: Liefert die Task des angegebenen Namens, falls sie existiert. Der eigene Katalog wird automatisch aktualisiert.

Fehlerfall:

* ... gibt es nicht

TASK PROC task (INT CONST channel number)

Zweck: Liefert den Namen der Task, die an dem angegebenen Kanal hängt.

Inter – Task – Kommunikation

Die Task – Kommunikation im EUMEL System ist strikt botschaftsorientiert. Eine Botschaft bzw. "Sendung" besteht immer aus einem Sendungscode (INT) und einem Datenraum (DATASPACE). Damit kann eine Botschaft bis zu 1 Mbyte umfassen!

Kommunikation zwischen zwei Tasks ist nur dann möglich, wenn Senders und Empfängers dazu bereit sind. Eine Sendung kann also nur dann korrekt transferiert werden, wenn der Empfänger existiert und empfangsbereit ist. Diese Art der Kommunikation wurde gewählt, um

- eine möglichst einfache und effiziente Implementation zu ermöglichen,
- mit den vorhandenen Primitiva möglichst flexibel bei der Implementation "höherer" Kommunikationsmethoden (z.B. Warteschlangen) zu sein.

call

PROC call (TASK CONST destination, INT CONST send code,
DATASPACE VAR message ds, INT VAR reply code)

Zweck: Die eigene Task wartet, bis die Zieltask 'destination' empfangsbereit ist. Dann wird die Sendung ('send code' und 'message ds') transferiert. Anschließend wartet die Sendertask auf eine Antwort von 'destination'. Für Sendungen anderer Tasks ist sie dabei nicht (!) empfangsbereit, nur die Zieltask kann eine Antwortsendung schicken. Nachdem eine solche Antwort eingetroffen ist, wird sie in 'message ds' und 'reply code' geliefert und die eigene Task fortgesetzt. Wenn die angesprochene Zieltask nicht existiert, wird – 1 als 'reply code' geliefert. 'message ds' ist in diesem Fall unverändert.
'call' hat Ähnlichkeiten mit einem Prozeduraufruf, nur ist es hier der Aufruf einer anderen Task. Störungen können hierbei nicht auftreten, da der Zustand der Zieltask keine Rolle spielt – es wird auf Empfangsbereitschaft gewartet – und beim Warten auf Antwort auch keine "Querschlägersendungen" von anderen Tasks dazwischenfunken können.

pingpong

PROC pingpong (TASK CONST destination, INT CONST send code,
DATASPACE VAR message ds, INT VAR reply code)

Zweck: Diese Prozedur wirkt wie die entsprechende 'call' – Prozedur, wartet aber nicht (!), bis die Zieltask empfangsbereit ist. Wenn die Zieltask existiert, aber nicht empfangsbereit ist, wird –2 als 'reply code' geliefert. Der 'message ds' ist dann nicht verändert.

send

PROC send (TASK VAR destination, INT CONST send code,
DATASPACE VAR message ds, INT VAR receipt)

Zweck: Wenn die Zieltask existiert und empfangsbereit ist, wird die Sendung ('send code' und 'message ds') transferiert und die Zieltask aktiviert. Als 'receipt' wird 0 (=ack) gemeldet. Diese positive Quittung kommt nicht von der Zieltask, sondern bestätigt nur, daß die Sendung ordnungsgemäß übertragen wurde. Der Datenraum gehört dann nicht mehr der Sender–, sondern der Zieltask, d.h. die Variable 'message ds' bezeichnet keinen gültigen Datenraum mehr.
Im Gegensatz zu 'call' und 'pingpong' läuft die Sendertask ohne Halt weiter und wartet nicht auf eine Antwort von der Zieltask.
Falls die Zieltask nicht existiert, wird –1, falls sie nicht empfangsbereit ist, –2 als 'receipt' geliefert. Bei diesen negativen Quittungen bleibt der Datenraum Eigentum der Absendertask, d.h. die Variable 'message ds' bezeichnet immer noch einen gültigen Datenraum.

PROC send (TASK VAR destination, INT CONST send code,
DATASPACE VAR message ds)

Zweck: s.o. Negative Quittungen (–1 oder –2) werden jedoch ignoriert. Der Datenraum wird entweder transferiert oder gelöscht ('forget'), steht also in keinem Fall mehr zur Verfügung. Die Prozedur sollte nur verwendet werden, wenn der Sender sicher ist, daß die Sendung transferiert werden kann, bzw. daß sie im Fehlerfall nicht transferiert zu werden braucht.

wait

PROC wait (DATASPACE VAR message ds, INT VAR message code,
TASK VAR source task)

Zweck: Die eigene Task geht in den offenen Wartezustand über. Sie ist jetzt gegenüber allen anderen Tasks empfangsbereit. Sie wird erst fortgesetzt, wenn eine Sendung eintrifft. Diese wird in 'message ds' und 'message code', die Absendertask in 'source task' geliefert.

Der Sendungscode muß zwischen den Beteiligten abgesprochen sein und ist also frei wählbar. Allerdings sind negative Werte nicht erlaubt, sondern für bestimmte "Pseudoantworten" vom Betriebssystem reserviert:

- –1 "Zieltask existiert nicht"
- –2 "Zieltask ist nicht empfangsbereit"
- –4 "Eingabe vom Kanal" Diese Meldung kann nur (!) beim offenen Warten ('wait') auftreten, und auch dann nur, wenn die Task gleichzeitig an einen Kanal angekoppelt ist. Hiermit wird mitgeteilt, daß mindestens ein Zeichen vorliegt. Dieses kann im folgenden mit 'inchar', 'incharety', 'blockin' oder darauf aufbauenden Prozeduren gelesen werden.

Weitere Codes werden in Systemroutinen standardmäßig verwandt und sollten auch von Anwenderroutinen genauso interpretiert werden:

0	"ack"	positive Quittung
1	"nak"	negative Quittung
2	"error nak"	negative Quittung mit Fehlermeldung. Der gelieferte Datenraum sollte die Struktur eines BOUND TEXTs haben und die Fehlermeldung in diesem TEXT beinhalten.

Beispiel: Kommunikation mit einem Manager

Auftraggeber	Manager
call (....)	REP wait (ds, order, order task) ; execute order ; send (order task, reply, ds) PER

Da der Auftraggeber 'call' verwendet, wartet er automatisch so lange, bis der Manager für ihn empfangsbereit wird. Dann schickt er die Sendung und geht gleichzeitig (!) in den geschlossenen "auf Antwort warten" – Zustand über. Der Manager kann daher unbesorgt mit dem "unsicheren" 'send' antworten, da die Empfangsbereitschaft des Auftraggebers nur durch Katastrophen wie Löschung der Task oder "halt from terminal" gestört werden kann. (In diesen Fällen kann die Antwort ruhig ins Leere gehen.)

Hier sieht man auch den Unterschied zwischen

call (...) und send (....); wait (....) .

Bei der zweiten Alternative können drei Störfälle eintreten:

a) Beim 'wait' kann eine Störsendung einer anderen Task eintreffen.

b) Da über die zeitlichen Rahmenbedingungen nichts ausgesagt werden kann, ist es möglich, daß der Manager die Antwort schickt, bevor die 'wait' – Operation beim Auftraggeber ausgeführt werden konnte. In unserem Beispiel würde das den Verlust der Rückmeldung und ewiges Warten seitens des Auftraggebers auslösen.

c) Der Manager ist nicht empfangsbereit. 'send' versagt, 'wait' wartet ewig.

2. Supervisor

Allgemein verfügbare Supervisor – Operationen

begin

PROC begin (PROC start, TASK VAR new task)

Zweck: Es wird eine unbenannte Task (Pseudoname " – ") als neuer Sohn der aufrufenden eingerichtet und mit der Prozedur 'start' gestartet. Namenskollision ist nicht möglich, die erzeugte Task kann aber auch nicht namensmäßig angesprochen werden. 'new task' identifiziert den neuen Sohn, falls kein Fehler auftrat.

Fehlerfälle :

* zu viele Tasks

PROC begin (TEXT CONST son name, PROC start, TASK VAR new task)

Zweck: Es wird eine Task mit Namen 'son name' als Sohn der aufgerufenen eingerichtet und mit der Prozedur 'start' gestartet. 'new task' identifiziert den neuen Sohn, falls kein Fehler auftrat.

Fehlerfälle :

* zu viele Tasks
* Name unzulaessig (* "" oder LENGTH > 100 *)
* ... existiert bereits

begin password

PROC begin password (TEXT CONST password)

Zweck: Diese Operation ist keine Supervisor – , sondern eine taskeigene Operation. Bei normalen 'global manager' – Tasks kann man so das weitere Kreieren von Sohntasks unter Paßwortkontrolle stellen. Wenn dieses Kommando in der Manager – Task gegeben worden ist, wird bei folgenden SV – begin – Kommandos interaktiv das Paßwort verlangt. Dabei gelten die üblichen Paßwort – Konventionen:

a) "" (Niltext) bedeutet *kein Paßwort*. Damit kann man durch 'begin password ("")' das Paßwort wieder ausschalten.
b) " – " bedeutet *jedes eingegebene Paßwort ist ungültig*. Damit kann man durch 'begin password (" – ")' das Einrichten von Sohntasks von außen (durch SV – Kommando) abschalten.

break

PROC break

Zweck: Die Task koppelt sich von einem evtl. angekoppelten Terminal ab. Bei der Abkopplung wird auf dem Terminal die "Tapete" ("Terminal n... EUMEL Version ..../M...") ausgegeben.

PROC break (QUIET CONST quiet)

Zweck: Die Task koppelt sich von einem evtl. angekoppelten Terminal ab. Dabei wird aber keine "Tapete" ausgegeben.

channel

INT PROC channel

Zweck: Liefert die Kanalnummer der eigenen Task. Falls kein Kanal (Terminal) zugeordnet ist, wird 0 geliefert.

INT PROC channel (TASK CONST task)

Zweck: Liefert die Kanalnummer der angegebenen Task. Ist kein Kanal zugeordnet, wird 0 geliefert.

clock

REAL PROC clock (INT CONST index)

Zweck: Liefert die über Index spezifizierte Systemuhr. Die Zeiteinheit ist 1 sec, die Meßgenauigkeit 0.1 sec.

clock (0) : CPU – Zeit der eigenen Task
clock (1) : Realzeit des Systems

REAL PROC clock (TASK CONST task)

Zweck: Liefert die CPU – Zeit der angegebenen Task.

Hinweis: Die CPU – Zeit beginnt mit der Taskkreation zu laufen. Sie gibt also jeweils die gesamte bisher verbrauchte CPU – Zeit an. Die Zeitdauer bestimmter Operationen kann als Differenz zweier 'clock' – Aufrufe gemessen werden. Beim Ende einer Task wird ihr CPU – Zeitverbrauch dem Vater zugeschlagen, um Abrechnungen zu ermöglichen.

continue

PROC continue (INT CONST channel nr)

Zweck: Die Task versucht, sich an den vorgegebenen Kanal anzukoppeln. Falls sie vorher schon an ein Terminal gekoppelt war, wird implizit 'break' durchgeführt, falls die Aktion erfolgreich durchgeführt werden konnte. Ein erfolgreiches 'continue' beinhaltet implizit 'reset autonom'.

Anmerkung: Normale Tasks können auf die Kanäle 1 – 24 zugreifen, Systemtasks dürfen sich auch an die privilegierten Kanäle 25 – 32 ankoppeln.

Fehlerfälle:

* ungueltiger Kanal
* Kanal belegt

end

PROC end

Zweck: Löscht die eigene Task und alle Söhne. Wenn die Task an ein Terminal angekoppelt ist, wird vorher angefragt, ob wirklich gelöscht werden soll. Anschließend wird die Standardtapete auf dem Bildschirm ausgegeben.

PROC end (TASK CONST task)

Zweck: Löscht die angegebene 'task'. 'task' muß allerdings die eigene oder eine Sohn – bzw. Enkel – Task der eigenen sein (siehe auch: 'Privilegierte Supervisor – Operationen'). Im Unterschied zur oben aufgeführten parameterlosen Prozedur 'end' wird nicht angefragt und auch keine Tapete ausgegeben. Wenn also die eigene Task ohne Reaktion auf dem Terminal beendet werden soll, kann dies mit 'end (myself)' geschehen.

Fehlerfall:

* 'end' unzulaessig

family password

PROC family password (TEXT CONST password)

Zweck: Diese Prozedur setzt oder ändert das Paßwort derjenigen Familienmitglieder, die kein Paßwort oder das gleiche Paßwort wie die aufrufende Task haben.

Zu einer Familie gehören die Task in der man sich befindet und die ihr untergeordneten Tasks.

Natürlich gelten auch hier die allgemeinen Paßwortbedingungen (siehe dazu: 'task password').

Beispiel: Das Kommando 'family password ("EUMEL")' wird in SYSUR gegeben. Dadurch wird das SYSUR-Paßwort und die Paßworte der entsprechenden Tasks unter SYSUR auf "EUMEL" gesetzt.

info password

PROC info password (TEXT CONST alt, neu) Zweck: Mit dieser Prozedur kann das Info-Paßwortes (max 9 Zeichen) gesetzt (alt = "") oder geändert werden.

Beachte: Anschließend wird automatisch ein 'shutup' durchgeführt. Außerdem gelten auch bei dieser Prozedur die allgemeinen Paßwortbedingungen (siehe dazu 'task password').

Hinweis: Wird als neues Paßwort "–" eingegeben, verliert der Urlader die Infofähigkeit.

next active

PROC next active (TASK VAR task)

Zweck: 'task' wird auf die nächste aktive Task gesetzt. Aktiv sind alle Tasks, die sich im Zustand 'busy' befinden oder auf Ein/Ausgabe warten (i/o) und an einen Kanal angekoppelt sind. Beispiel:

```
TASK VAR actual task := myself;
REP
  ... ;
  next active (actual task)
UNTIL actual task = myself PER.
```

Hier werden alle aktiven Tasks durchgemustert (z.B. für Scheduling–Anwendungen). Dieses Verfahren ist sehr viel weniger aufwendig als eine Durchmusterung des ganzen Taskbaumes, liefert aber nur die gerade aktiven Tasks.

rename myself

PROC rename myself (TEXT CONST new task name)

Zweck: Die eigene Task erhält als neuen Tasknamen 'new task name'. Damit kann auch aus einer benannten eine unbenannte Task mit dem Pseudonamen "–" werden. Umbenennung in die andere Richtung ist ebenfalls möglich.

Achtung: Durch das Umbenennen der Task werden alle Taskvariablen, die sich auf diese Task beziehen, ungültig (als wäre die Task gelöscht und dann neu eingerichtet).

Fehlerfälle:

* ... existiert bereits
* Name unzulaessig

reset autonom

PROC reset autonom

Zweck: Die eigene Task deklariert sich beim Supervisor als nicht autonom (Normalzustand). Das bedeutet, 'continue' – Aufforderungen über ein 'Supervisor – Kommando' vom Terminal werden vom System ohne Benachrichtigung der Task durchgeführt.

set autonom

PROC set autonom

Zweck: Die eigene Task deklariert sich beim Supervisor als autonom (üblich für Manager – Tasks). Wenn jetzt ein 'continue' – Supervisor – Kommando auf diese Task von einem Terminal aus gegeben wird, wird der Task über 'send' eine Nachricht zugestellt.

Achtung: Im autonomen Zustand ist der Programmierer selbst für die Reaktion der Task verantwortlich. Man kann sie von außen auf keine Weise gewaltsam an ein Terminal koppeln (ermöglicht Paßalgorithmen / Datenschutz).

Um die Programmierung etwas zu entschärfen, wird eine Task automatisch aus dem autonomen in den Normalzustand überführt, wenn sie selbst ein 'continue' gibt (s. dort).

status

INT PROC status (TASK CONST task)

Zweck: Liefert den Status der angegebenen Task:

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, ist 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.

storage

PROC storage (INT VAR size, used)

Zweck: Informiert über den physisch verfügbaren ('size') und belegten ('used') Speicher des Gesamtsystems. Die Einheit ist KByte.

Achtung: 'size' gibt den Speicher an, der benutzt werden kann, ohne in eine Engpaßsituation zu kommen. Tatsächlich wird auf dem Hintergrundmedium noch eine gewisse Reserve freigehalten. Wenn diese angebrochen wird, befindet sich das System im Speicherengpaß. Dieser Zustand kann mit 'used > size' abgefragt werden.

INT PROC storage (TASK CONST task)

Zweck: Liefert die Größe des Speicherbereichs in KByte, den die angegebene Task augenblicklich belegt.

Dabei werden durch Sharing mögliche Optimierungen nicht berücksichtigt. D.h. eine Task kann physisch erheblich weniger Speicher als logisch belegen. Entsprechend kann die Speichersumme aller Tasks den physisch belegten Speicherbereich des Gesamtsystems beträchtlich überschreiten.

task password

PROC task password (TEXT CONST password)

Zweck: Das angegebene Paßwort wird beim Supervisor hinterlegt. Bei folgenden SV – Kommandos 'continue...' auf diese Task wird interaktiv das Paßwort abgefragt. Dabei gelten die üblichen Paßwort – Konventionen:

a) "" (Niltext) bedeutet *kein Paßwort.* Damit kann man durch 'task password ("")' das Paßwort wieder ausschalten.

b) "–" bedeutet *jedes eingegebene Paßwort ist ungültig.* Damit kann man durch 'task password ("–")' das Ankoppeln an ein Terminal von außen (durch SV – Kommando) unterbinden.

Privilegierte Supervisor – Operationen

Die im folgenden aufgeführten privilegierten Operationen können nur von Systemtasks – das sind direkte oder indirekte Söhne des Supervisors – ausgeführt werden. Um Mißbrauch unmöglich zu machen, sollte der Supervisor nach der Einrichtung der gewünschten Systemtasks bzgl. der Einrichtung neuer Söhne gesperrt und alle Systemtasks durch Paßworte geschützt werden (siehe Teil 1).

block

PROC block (TASK CONST task)

Zweck: Die angegebene Task wird blockiert, d.h. so lange von der Verarbeitung suspendiert, bis die Blockade durch 'unblock' wieder aufgehoben wird. Diese Operation wird vom Scheduler benutzt. Falls das Packet 'scheduler' insertiert ist, sollten andere Tasks die Prozedur 'block' nicht anwenden, um dem Scheduling nicht entgegenzuwirken.

collect garbage blocks

PROC collect garbage blocks

Zweck: Es wird eine außerplanmäßige Gesamtmüllabfuhr durchgeführt. Planmäßig (d.h. ohne Aufruf dieser Prozedur) wird sie alle 15 Minuten und in Engpaßsituationen durchgeführt. Nach Aufruf dieser Prozedur wird der automatische Fixpunkt/ Müllabfuhr – Rhythmus ca. 1 Stunde lang gesperrt. Somit kann man z.B. in der Task "scheduler" einen eigenen Fixpunkt/Müllabfuhr – Rhythmus implementieren.

Achtung: Diese Operation erfordert starkes Paging und dauert dementsprechend lange.

end

PROC end (TASK CONST task)

Zweck: Die angegebene Task und alle Söhne, Enkel etc. werden gelöscht. Systemtasks (direkte und indirekte Nachkommen des SUPERVISORs) können beliebige andere Tasks (nicht nur eigene Söhne) löschen.

fixpoint

PROC fixpoint

Zweck: Für das Gesamtsystem wird ein außerplanmäßiger Fixpunkt geschrieben. Planmäßige Fixpunkte (d.h. ohne Aufruf dieser Prozedur) werden alle 15 Minuten geschrieben. Nach Aufruf dieser Prozedur wird der automatische Fixpunkt/Müllabfuhr – Rhythmus ca. 1 Stunde lang gesperrt. Somit kann man z.B. in der Task "scheduler" einen eigenen Fixpunkt/Müllabfuhr – Rhythmus implementieren.

Achtung: Diese Operation verursacht starkes Paging (Rückschreiben aller veränderten Seiten auf das Hintergrundmedium) und dauert dementsprechend lange.

prio

INT PROC prio (TASK CONST task)

Zweck: Liefert die augenblickliche Priorität der angegebenen Task.

PROC prio (TASK CONST task, INT CONST new prio)

Zweck: Setzt die Priorität der Task.

Hinweis: 0 ist die höchste, 15 die niedrigste Priorität. Die Prioritäten 0 bis 2 werden von EUMEL 0 (fine scheduling) verwaltet. Die restlichen Prioritäten können für 'rough scheduling' (siehe auch im Kapitel Scheduler) eingesetzt werden.
Durch 'continue ("name")' wird die Priorität wieder auf 0 gesetzet.

set date

PROC set date

Zweck: Datum und Uhrzeit können im Dialog gesetzt werden (Form wie beim Start des Systems). Dabei wird gegebenenfalls die Hardware-Uhr gelesen.
Sollte der SHard ein falsches Datum liefern, so muß das Datum mit

```
set clock (date("tt.mm.jj") + time ("hh:mm:ss")
```

gesetzt werden.

save system

PROC save system

Zweck: Der gesamte Systemhintergrund wird auf Archivdisketten gesichert. Zu diesem Zweck wird das System wie bei 'shutup' heruntergefahren.

shutup

PROC shutup

Zweck: Kontrolliertes Herunterfahren des Systems. Beim nächsten Systemstart wird automatisch Datum und Uhrzeit erfragt, wenn der Kommandodialog eingeschaltet ('command dialogue (TRUE)') und keine Hardwareuhr vorhanden ist. Falls diese Prozedur nicht vor dem Abschalten aufgerufen wurde, findet beim Neustart ein Aufsetzen auf dem letzten Fixpunkt statt (RERUN).

unblock

PROC unblock (TASK CONST task)

Zweck: Eine vorherige Blockierung der Task wird aufgehoben. Ist die Task nicht blockiert, bewirkt 'unblock' nichts. Diese Operation wird vom Scheduler benutzt. Andere Tasks sollten sie normalerweise nicht anwenden, um dem Scheduling nicht entgegenzuwirken.

3. ID - Konstanten

Die Informationsprozedur

```
INT PROC id (INT CONST no)
```

liefert folgende Informationen über die Soft- und Hardware des Rechners:

Von EUMEL 0 werden geliefert:

```
id (0) - - > EUMEL-Version
id (1) - - > Prozessortyp    ( 1: Z80;
                               2: Z8001;
                               3: 8086 und kompatible;
                               4: 68000 )
id (2) - - > Urlader-Version
id (3) - - > reserviert
```

Vom SHard werden geliefert:

```
id (4) - - > Lizenznummer des SHards
id (5) - - > Installationsnummer des EUMEL-Anwenders
id (6) - - > SHard-spezifisch
id (7) - - > SHard-spezifisch
```

4. Systemverwaltung

Achtung: Dieser Teil des Systemhandbuchs ist nur für solche Multi – User – Installationen von Bedeutung, die erweiterte Systemverwaltungsfunktionen generieren bzw. modifizieren wollen.

Das EUMEL – System ist in der ausgelieferten minimalen Standardform (ohne die Features) ohne weiteres benutzbar.

Der Systemmanager SYSUR

Der Systemmanager verhält sich im wesentlichen wie ein normaler, allerdings mit folgender Erweiterung:

- Die Operationen 'list' und 'fetch' können von allen Tasks des Systems und nicht nur von Söhnen durchgeführt werden. Damit kann man Systemverwaltungsdateien (z.B. "logbuch") von allen Tasks aus lesen. 'erase' und 'save' sind jedoch nur von Söhnen bzw. Enkeln – d.h. von privilegierten Systemtasks – aus zulässig.

Das Paket stellt folgende Operationen zusätzlich zur Verfügung:

generate shutup manager

PROC generate shutup manager

Zweck: Es wird eine Sohntask mit Namen "shutup" kreiert. Diese Task ist nicht (!) paßwortgeschützt, läßt aber keine normalen Kommandos zu, sondern fragt nur

shutup (j/n) ?

So kann jeder das System kontrolliert abschalten und die privilegierten Operationen des OPERATORs wie 'end' sind dennoch geschützt.

put log

PROC put log (TEXT CONST log record)

Zweck: Der angegebene 'log record' wird mit vorangestelltem Tasknamen des Absenders, Datums- und Uhrzeitangabe in die Logbuchdatei "logbuch" in der Task "SYSUR" geschrieben. Der neue Satz wird an die Datei angefügt. ("logbuch" wird z.B. vom EUMELmeter verwandt.)

Hinweis: Bei Verwendung des Logbuchs darf die zwar große, aber doch endliche Dateikapazität nicht vergessen werden. Nachdem das Logbuch mit 4073 Sätzen voll ist, werden weitere 'put log' Operationen ignoriert. Die Datei "logbuch" sollte deshalb – wenn sie beispielsweise vom EUMELmeter verwandt wird – von Zeit zu Zeit gelöscht werden ('erase' bzw. 'forget')!

Scheduler

Der Scheduler dient zur Verwaltung der rechenwilligen Hintergrundtasks. Will man den Scheduler (eventuell abgeändert) insertieren, muß man die Task "scheduler" als Sohn von SYSUR einrichten. Dann holt man die Datei "scheduler" vom Archiv und insertiert sie. "scheduler" beinhaltet "eumelmeter". Es wird beim Start erfragt, ob die Meßroutinen aktiviert werden sollen oder nicht.

Funktionsweise des Schedulers

Der Scheduler sammelt in bestimmten Zeitintervallen alle aktiven (rechnenden) Tasks ab, die an kein Terminal angekoppelt sind und auch keine Manager sind. Diese Tasks werden blockiert und in die Warteschlange der Standardklasse eingefügt.

Die Klassen des Schedulers werden durch die Taskprioritäten 5 bis 9 definiert. Die Standardklasse entspricht der Priorität 7. Die Klassenzugehörigkeit einer Task kann von einer Systemtask aus (z.B. von "OPERATOR") mit der Prozedur 'prio' verändert werden.

Der Scheduler geht nach folgender Strategie vor:

Anhand der Vordergrund/Hintergrundlast des Systems wird entschieden, ob überhaupt Hintergrundtasks aktiv sein dürfen, welche Klassen aktiv sein dürfen und wieviel Hintergrundtasks gleichzeitig rechnen dürfen.

Die wartenden Hintergrundtasks werden im Round – Robin – Verfahren aktiviert. Dabei kommt die Klasse n + 1 erst dann zum Zug, wenn die Warteschlange der Klasse n leer ist oder weniger Tasks enthält, als gleichzeitig aktiviert werden sollen.

Die implementierte Standardstrategie hat als oberste Maxime, den Vordergrund auf keinen Fall zu stören. Dementsprechend wird der Hintergrund nur aktiviert, wenn eine der folgenden Bedingungen erfüllt ist:

- Die Vordergrundlast des Systems liegt unter 3% .

- Es ist keine normale Vordergrundtask (Nachfahre von "UR") an einen Kanal angekoppelt. Man beachte, daß Systemtasks hierbei nicht berücksichtigt werden. Ein aktiver Drucker blockiert die Hintergrundtasks also nicht.

EUMELmeter (Systemstatistik)

Die Meßsoftware zum Protokollieren der Systembelastung befindet sich auf dem Archiv 'std.zusatz'.

Falls das System keinen Scheduler benutzt, muß eine Meßtask als Sohn von "SYSUR" eingerichtet werden. In diese Task muß dann die Datei "eumelmeter" vom Archiv gebracht und übersetzt werden.

Falls das System einen Scheduler beinhalten soll, muß bei der Generierung des Schedulers lediglich auf die Frage "mit eumelmeter (j/n) ?" mit "j" geantwortet werden.

EUMELmeter

Das EUMELmeter protokolliert die Systemlast in ca. 10 minütigen Abständen in der Datei "logbuch" in "SYSUR". Für jedes Meßintervall wird eine Zeile angefügt. Die Zeilen sind folgendermaßen aufgebaut:

```
tt.mm.jj hh:mm    hg  uf ub   pw pb  cpuf cpub cpus last nutz
```

tt.mm.jj hh:mm	Datum und Uhrzeit des Eintrags
hg	Größe des aktuell belegten Hintergrundspeichers (in KB)
uf	Anzahl der aktiven Vordergrundtasks
ub	Anzahl der aktiven Hintergrundtasks
pw	Paginglast bei wartender CPU (Paging/Wait)
pb	Paginglast bei aktiver CPU (Paging/Busy)
cpuf	CPU – Auslastung durch Vordergrundtasks
cpub	CPU – Auslastung durch Hintergrundtasks
cpus	CPU – Systemlast
last	Gesamtlast des Systems: pw + pb + cpuf + cpub + cpus (Achtung: kann 100% übersteigen, da Platte und CPU überlappt arbeiten können.)
nutz	Nutzgüte im Meßintervall: 100% – pw – cpus Die Nutzgüte gibt an, welcher Anteil der Systemarbeit für echte Nutzarbeit verfügbar war. Sie ist die Summe aus der echten Nutzlast 'cpuf + cpub' und der Leerzeit, die ja theoretisch auch für Nutzarbeit hätte verwandt werden können. Sie läßt sich, wie oben angegeben, auch berechnen, indem man den idealerweise überflüssigen Overhead 'cpus' und 'pw' von 100% abzieht.

TEIL 6: Der EUMEL – Drucker

Die Ansteuerung eines Druckers durch das EUMEL – System geschieht durch zwei aufeinanderbauende Komponenten. Die eine Komponete ist der hardwareunabhängige EUMEL – Drucker, der die Textverarbeitungsanweisungen umsetzt und die Druckseite entsprechend aufbereitet, so daß sie im Blocksatz, in Tabellenform oder in Spalten gedruckt werden kann. Die andere Komponente ist der hardwareabhängige Druckertreiber, der durch ein einfaches Interface zum EUMEL – Drucker, wie z.B. Textausgeben, Positionieren oder Schrifttypen und Modifikationen an – und ausschalten, den eigentlichen Druck vornimmt.
Die hardwareunabhängige Komponente, der EUMEL – Drucker, befindet sich bei den ausgelieferten Systemen im priviligierten Ast des Taskbaums, so daß beim Anschluß eines Druckers nur noch der hardwareabhängige Druckertreiber insertiert werden muß. Auf dem Standardarchiv "std.printer" befinden sich schon einige Druckeranpassungen für diverse Druckertypen (siehe Teil 1, Kapitel "Druckersofware im Multi – User").
Bei Single – User Systemen, die über ausreichend Hintergrundkapazität verfügen (ab 720K), kann der EUMEL – Drucker nachinsertiert werden. Er wird auf der Archivdiskette "std.zusatz" ausgeliefert. Soll nun ein neuer Drucker angeschlossen werden, so sind die folgenden Schritte zu unternehmen.

- Implementierung des Druckertreiber – Interface
  (siehe Kapitel 1: "Das Druckertreiber – Interface")
  Das Paket mit dem Druckertreiber muß in einer Task "PRINTER" insertiert und ein Spool eingerichtet werden.

- Erstellen einer Fonttabelle für den anzuschießenden Drucker
  (siehe Teil 7: "Der Fontspeichers")
  Nach der Erstellung der Fonttabelle muß diese in die Task "configurator" gebracht werden. Die Fonttabelle sollte in allen bestehenden Tasks – insbesondere in der Task "PUBLIC" und der Task "PRINTER" – mit dem ***fonttable*** – Kommando eingestellt werden.

1. Das Druckertreiber – Interface

Da der EUMEL – Drucker vor dem Druckertreiber insertiert ist, aber auf dem Druckertreiber aufbaut, müssen beim Aufruf der 'print' – Prozedur des EUMEL – Druckers die Prozeduren des Druckertreibers mit übergeben werden. Aus progammtechnischen Gründen sollte ihre Anzahl möglichst gering gehalten werden. Deshalb gibt es die folgende drei Prozeduren, die mit einem 'op code' parametrisiert werden. Die Bedeutung der weiteren Parameter der Interfaceprozeduren hängen von diesem 'op code' ab. Die folgende Beschreibung der Prozeduren gibt einen Programmrahmen vor, in dem die Parameter durch Refinements entsprechend ihrer Bedeutung umdefiniert sind.

```
PROC open (INT CONST op code, INT VAR param 1, param 2) :

  LET document code  =  1 ,
      page code      =  2 ;

  SELECT op code OF
    CASE document code  :  open document
    CASE page code      :  open page
  END SELECT.


  x steps : paraml .
  y steps : param2 .

  open document :
```

Zweck: Die Prozedur wird vom EUMEL – Drucker zur Einleitung jedes Ausdrucks aufgerufen. Hier können notwendige Initialisierungen der Hardware durchgeführt werden. In 'x steps' und 'y steps' muß die Breite bzw. Höhe der bedruckbaren Fläche des Papieres in Mikroschritten des Druckers angegeben werden.[1)]

```
  x start : paraml .
  y start : param2 .
```

1) Zur Definition der Mikroschritte siehe Bemerkung 2.

```
  open page :
```
Zweck: Hiermit wird dem Hardware – Interface der Beginn einer neuen Seite mitgeteilt. Die Parameter 'x start' und 'y start' liefern die gewünschte Position der linken oberen Ecke des Schreibfeldes. Das Hardware – Interface muß in diesen Parametern seine augenblickliche Position auf dem Papier zurückmelden, wobei die Position (0,0) die linke obere Ecke des Papieres ist.
Vor der Rückmeldung kann aber auch auf die angegebene Startposition positioniert und diese zurückgemeldet werden. Es ist jedoch darauf zu achten, daß die zurückgemeldete Position den internen Nullpunkt für die Absolutkoordinaten im EUMEL – Drucker definiert. Deswegen muß das Hardware – Interface sicherstellen, daß bei einem "Zeilenrücklauf" die zurückgemeldete Position 'x start' erreicht wird. (Siehe 'carriage return' in der 'PROC execute'). Auch gibt es Fälle, bei denen links von der gemeldeten 'x start' – Position positioniert wird.
Bei Druckern mit Einzelblatteinzug, bei denen das Papier gleich auf die zweite oder dritte Zeile positioniert wird, sollte, um ein korrektes Druckbild zu erreichen, diese Postion in 'y start' zurückgemeldet werden.

```
END PROC open;


PROC close (INT CONST op code, INT CONST param 1) :

  LET document code  =  1 ,
      page code      =  2 ;

  SELECT op code OF
    CASE document code  :  close document
    CASE page code      :  close page
  END SELECT.


  close document :
```
Zweck: Hiermit wird dem Hardware – Interface das Ende eines Druckvorgangs mitgeteilt.

```
  remaining y steps : param 1 .
```

```
  close page :
```

Zweck: Hiermit wird dem Hardware – Interface mitgeteilt, daß der Druck der aktuellen Seite abgeschlossen ist.
'remaining y steps' gibt an, wieviel Mikroschritte das vertikale Papierende noch von der aktuellen Druckposition entfernt ist. Die x – Position des Druckers ist bei Aufruf dieser Prozedur immer der linke Rand 'x start'.

```
END PROC close;


PROC execute (INT CONST op code, TEXT CONST string,
                                 INT  CONST param1, param2) :

  LET write text code       =  1 ,
      write cmd code        =  2 ,
      carriage return code  =  3 ,
      move code             =  4 ,
      draw code             =  5 ,
      on code               =  6 ,
      off code              =  7 ,
      type code             =  8 ;

  SELECT op code OF
    CASE write text code       :  write text
    CASE write cmd code        :  write cmd
    CASE carriage return code  :  carriage return
    CASE move code             :  move
    CASE draw code             :  draw
    CASE on code               :  on
    CASE off code              :  off
    CASE type code             :  type
  END SELECT .


  from : param1 .
  to   : param2 .
```

```
write text :
```
Zweck: Der übergebene Text 'string' muß von der Zeichenposition 'from' bis 'to' (einschließlich) auf dem Drucker ausgegeben werden. Die Überschreitung der Papierbreite braucht nicht überprüft zu werden.

```
write cmd :
```
Zweck: Der übergebene Text 'string' enthält zwischen den Positionen 'from' und 'to' ein direkt angegebenes Druckerkommando (#"..."#). Wenn direkte Druckerkommandos erlaubt sein sollen, müssen sie ausgegeben werden.

```
x steps to left margin : param 1 .
```

```
carriage return :
```
Zweck: Der Druckkopf muß (ohne Zeilenvorschub) an den linken Rand bewegt werden, d.h. an die bei 'open page' vom Druckertreiber gemeldete Position 'x start'. 'x steps to left margin' gibt an, wieviel Minimalschritte die augenblickliche Position vom linken Rand entfernt ist.

```
x steps : param 1 .
y steps : param 2 .
```

```
move :
```
Zweck: Die Schreibposition muß um 'x steps' Mikroschritte nach rechts und um 'y steps' Mikroschritte nach unten verschoben werden. Negative Schrittwerte bedeuten dabei die jeweils andere Richtung. Das Überschreiten des Papiers braucht nicht überprüft zu werden. Bei einer horizontalen Bewegung nach rechts ('x steps' > 0) müssen die eingeschalteten Modifikationen beachtet werden. Wenn z.B. 'underline' eingeschaltet ist, muß die Strecke unterstrichen werden.
Kann eine Leistung (z.B. Bewegung nach links) nicht erbracht werden, muß ein 'errorstop' ausgelöst werden. Im Fehlerfall bei einer Horizontalbewegung versucht der EUMEL – Drucker nach einem Zeilenrücklauf nochmals die angestrebte x – Position zu erreichen. Somit brauchen horizontale Bewegungen nach links ('x steps' < 0) nicht unbedingt implementiert zu werden, sondern können mit einem 'errorstop' beantwortet werden. Im Fehlerfall bei einer vertikalen Bewegung wird an der alten Position weitergeschrieben.

```
draw :
```

Zweck: Von der aktuellen Schreibposition an (linke untere Ecke der Zeichenposition) soll eine gerade Linie zum Zielpunkt ('x steps' weiter rechts, 'y steps' weiter unten) gezogen werden. Kann eine Leistung (z.B. schräge Linie, Linie nach oben o.ä.) nicht erbracht werden, muß ein 'errorstop' ausgelöst werden. Dieser Fehlerfall wird vom EUMEL – Drucker ignoriert. Das Überschreiten des Schreibfeldes braucht nicht überprüft zu werden.

```
modification : param 1 .
```

```
on :
```

Zweck: Die Modifikation der Nummer 'modification' soll eingeschaltet werden, sofern die Hardware es erlaubt. Augenblicklich gibt es folgende Modifikationen:

1	underline
2	bold
4	italics
8	reverse

Die in der Fonttabelle spezifizierte Befehlssequenz, um die entsprechende Modifikation anzuschalten, kann mit der Prozedur *on string (modification)* abgefragt werden.
Kann eine Leistung nicht erbracht werden, muß ein 'errorstop' ausgelöst werden. Im Fehlerfall der Modifikation 'underline' versucht der neue EUMEL – Drucker die Zeile mit Hilfe von 'draw' zu unterstreichen. Im Fehlerfall der Modifikation 'bold' wird die Zeile nochmals, um den in der Fonttabelle spezifizierten 'bold offset' verschoben, ausgegeben. Bei den restlichen beiden Modifkationen wird der Fehlerfall ignoriert.

```
off :
```

Zweck: Die angegebene Modifikation 'modification' soll ausgeschaltet werden. Die in der Fonttabelle spezifizierte Befehlssequenz, um die entsprechende Modifikation auszuschalten, kann mit der Prozedur *off string (modification)* abgefragt werden. Ein Fehlerfall wird hier ignoriert.

```
font nr : param 1 .
```

```
type :
```

Zweck: Die Druckausgabe soll auf den Schrifttyp mit der angegebenen Fontnummer 'font nr' umgeschaltet werden. Diese Nummer bezieht sich auf die eingestellte Fonttabelle. Mit den Prozeduren des Fontspeichers können anhand dieser Nummer die nötigen Informationen beschafft werden. So liefert z.B. die Prozedur *font string (font nr)* die in der Fonttabelle spezifizierte Befehlssequenz oder die Prozedur *font (font nr)* den Namen des Fonts. Die Breite des Leerzeichens kann mit *char pitch (font nr, " ")* bestimmt werden.

```
END PROC execute;
```

2. Prozedur – Schnittstelle des EUMEL – Druckers

print

```
PROC print  (PROC (TEXT VAR) next line, BOOL PROC eof,
             PROC (INT CONST, INT VAR, INT VAR) open,
             PROC (INT CONST, INT CONST) close,
             PROC (INT CONST, TEXT CONST,
                                INT CONST, INT CONST) execute,
             BOOL CONST elan listing, TEXT CONST file name)
```

Zweck: Solange die Prozedur 'eof' FALSE liefert wird mit der Prozedur 'next line' eine Zeile eingelesen. Dieser Eingabestrom wird für den Druck aufbereitet. Ist die Konstante 'elan listing' auf FALSE gesetzt, so wird der Eingabestrom als Textdatei mit Textkosmetik – Anweisungen ausgedruckt. Andernfalls wird der Eingabestrom wie ein ELAN – Listing behandelt. In der Textkonstanten 'file name' muß dann der Dateiname der Programmdatei enthalten sein.

```
PROC print (FILE VAR file,
             PROC (INT CONST, INT VAR, INT VAR) open
             PROC (INT CONST, INT CONST) close
             PROC (INT CONST, TEXT CONST,
                                INT CONST, INT CONST) execute)
```

Zweck: Der Eingabestrom kommt aus der angegebenen Datei. Anhand vorgegebener Kriterien wird entschieden, ob diese Datei als Textdatei oder als ELAN – Listing ausgedruckt wird.

with elan listings

```
PROC with elan listings (BOOL CONST flag)
```

Zweck: Mit dieser Prozedur kann bei der vorangegangenen 'print' – Prozedur gesteuert werden, ob überhaupt irgendwelche Dateien als ELAN – Listings gedruckt werden sollen. Wird damit FALSE eingestellt, so werden alle Dateien als Textdateien gedruckt.

```
BOOL PROC with elan listings
```

Zweck: Liefert die aktuelle Einstellung.

is elan source

PROC is elan source (FILE VAR file)

Zweck: Entscheidet nach vorgebenen Kriterien, ob die angegebene Datei ein ELAN – Listing ist.

bottom label for elan listings

PROC bottom label for elan listings (TEXT CONST label)

Zweck: Bei ELAN – Listings wird in der Fußzeile ein Text eingestellt, der durch Schägstrich getrennt vor die Seitennummer geschrieben wird. (z.B. zur Durchnumerierung der Pakete im Quellcode)

TEXT PROC bottom label for elan listings

Zweck: Liefert die aktuelle Einstellung.

material

TEXT PROC material

Zweck: Hier kann das Hardware – Interface jeder Zeit den aktuellen Materialwert abfragen, der vom Benutzer mit der 'material' – Anweisung eingestellt ist.

x pos

INT PROC x pos

Zweck: Wird in der Prozedur 'execute' die Funktion 'move' oder 'draw' angesteuert, so liefert diese Prozedur die absolute Zielposition in x – Richtung, wo bei der Nullpunkt durch das zurückgelieferte 'x start' bei 'open page' definiert ist. Diese Prozedur dient zur Unterstützung von Druckern, die eine absolute Positionierung in horizontaler Richtung benötigen.

y pos

INT PROC y pos

Zweck: Wird in der Prozedur 'execute' die Funktion 'move' oder 'draw' angesteuert, so liefert diese Prozedur die absolute Zielposition in y – Richtung, wo bei der Nullpunkt durch das zurückgelieferte 'y start' bei 'open page' definiert ist. Diese Prozedur dient zur Unterstützung von Druckern, die eine absolute Positionierung in vertikaler Richtung benötigen.

linetype

INT PROC linetype

Zweck: Wird in der Prozedur 'execute' die Funktion 'draw' angesteuert, so gibt diese Prozedur den gewünschten Linientyp an. Bisher ist nur definiert:

1 underline

Anmerkung: Bis jetzt benutzt der EUMEL – Druckers die Funktion 'draw' lediglich zum Unterstreichen in Fehlerfall der Modifikation 'underline', d.h. zeichnen mit 'y steps = 0' und 'x steps > = 0' mit 'line type = 1' reicht aus.

y offset index

INT PROC y offset index

Zweck: Wurde der Font mit 'y offsets' definiert, so kann hiermit in der bei der Funktion 'write text' in der Prozedur 'execute' der jeweilige Offset – Index für den auszugebenden Text abgefragt werden. Der Offset – Index sagt aus, die wievielte Verschiebung nun ausgegeben wird. Dabei werden die Verschiebungen in der Reihenfolge durchnummeriert, in der sie in der Fonttabelle angegeben wurden. Anhand dieses Offset – Index muß das Hardware – Interface entscheiden, welche Bitmuster ausgegeben werden müssen.

pages printed

INT PROC pages printed

Zweck: Gibt nach dem Ausdruck an, wieviel Seiten gedruckt wurden.

3. Bemerkungen und Ratschläge

1) Für ein Paket, das dieses Interface implementiert, sind folgende Punkte wichtig:

   - Man braucht sich keine Zustände (aktuelle Position o.ä.) zu merken.

   - Rückmeldungen über die Leistungsfähigkeit eines Druckers bzw. seiner Anpassung erfolgen über 'errorstop'. Der EUMEL – Drucker stellt fest, ob bestimmte Leistungen (Einschalten der Attribute und Bewegungen des Drukkers) verfügbar sind, indem er sie versuchsweise ausführen läßt. Bei den Prozeduren 'open', 'close' und den Funktionen 'write text', 'write cmd', 'carriage return' und 'type' der Prozedur 'execute' führt ein 'errorstop' jedoch zum Abbruch des Drucks.

2) Die ***Mikroschritte*** sollten die kleinsten durchführbaren horizontalen bzw. vertikalen Bewegungen des Druckers sein. Oft gibt aber das Handbuch des Druckers keine eindeutige Angabe über die Mikroschritte in horizontaler Richtung, sondern sagt nur, daß es gewisse Schriften mit einer bestimmten Anzahl von Zeichen pro Zoll gibt.[1] Dann ergibt sich die Anzahl von Mikroschritten pro Zoll aus dem kleinsten gemeinsamen Vielfachen der Anzahl Zeichen pro Zoll aller Schriften.

   Beispiel:

   Der Olivetti Drucker PR1470 hat drei Schriften mit 10, 12, und 16.6 Zeichen pro Zoll. Das kleinste gemeinsame Vielfache ist 300. Ein Mikroschritt bei dem Drucker PR1470 entspricht also einem 300stel Zoll. Die Breite der einzelnen Schriften läßt sich nun aus der folgenden Tabelle ablesen.

Anzahl Zeichen pro Zoll	Breite in 1/300 Zoll
10	30
12	25
16.6	18

   Wenn der Drucker in diesen theoretischen Mikroschritten nicht positionieren kann, so muß er bei einem *move* – Befehl so genau wie möglich positionieren. Der Rest sollte abgespeichert und beim nächsten *move* – Befehl hinzuaddiert werden.

---

1) 1 Zoll = 1 Inch = 2.54 cm

3) Um ein optimales Druckbild zu bekommen, müssen alle Breiten und Höhenangaben der Zeichen genau angegeben werden.

4) Die Fonttabelle bietet eine einfache Möglichkeit, Zeichen mit Hilfe der Ersatzdarstellungen umzucodieren. Deshalb sollte der Druckerkanal auch mit der Konfigurationstabelle 'transparent' konfiguriert werden.

5) Um den Schrifttyp festzulegen, mit dem ELAN – Listings gedruckt werden sollen, kann in der Fonttabelle einem Font der Name ***"elanlist"*** zugeordnet werden, denn der ELAN – Lister versucht auf einen Schrifttyp mit diesem Namen zuschalten. Wenn kein Schrifttyp "elanlist" existiert, dann wird für ELAN – Listings der erste Schrifttyp der Fonttabelle genommen.

6) Nach der Installation des Druckertreibers ist darauf zu achten, daß in der Task "PRINTER" eine Fonttabelle des Druckers eingestellt ist.

7) Der Druckertreiber sollte eventuell noch ein Prozedur bereitstellen, mit der die Papierbreite bzw. – höhe eingestellt werden kann, die bei 'open document' dem EUMEL – Drucker gemeldet wird.

4. Arbeitsweise des EUMEL – Druckers

Der EUMEL – Drucker arbeitet mit der folgenden Strategie:

Die Datei wird zeilenweise analysiert. Bei der Analyse werden einzelne Token bestimmt. Ein Token ist ein Textteil, der zusammenhängend gedruckt werden kann, ohne daß es zu Typumschaltungen, Modifkationsänderungen oder Positionierungen in x – bzw. y – Richtung kommt. So ist bei einfachem Zeilendruck jede Zeile ein Token, während im Blocksatz jedes Wort ein Token ist. Ein Token hat also immer

- einen Text,
- die Länge des Textes bei der Ausgabe,
- eine absolute x – und y – Position auf dem Papier,
- einen Schrifttyp,
- Modifikationen für den Text,
- Modifikationen für den Zwischenraum vom letzten Token zu diesem Token.

Sind alle Token einer Zeile bestimmt, so werden sie in eine Liste aller bisher erzeugten, aber noch nicht gedruckten Token der absoluten y – Position nach einsortiert. Diese Tokenliste wird erst dann ausgedruckt, wenn sichergestellt ist, daß im weiteren Verlauf der Datei kein Token vor das letzte Token der sortierten Liste kommt. Beim Zeilendruck ist dies nach jeder Zeile der Fall. Bei Spaltendruck kann jedoch erst dann ausgedruckt werden, wenn sich die Analyse in der letzten Spalte befindet. Spätestens bei einem Seitenwechsel muß die Tokenliste ausgegeben werden.

Durch diese Strategie lassen sich Spaltendruck oder Indizes und Exponenten sehr leicht für alle Drucker implementieren, ohne daß ein Drucker in vertikaler Richtung rückwärts positionieren muß.

Bei der Ausgabe der Tokenliste wird jeweils auf die nächst größere y – Position positioniert und dort werden alle Token zu dieser y – Position ausgegeben. Die Ausgabe eines Tokens erfolgt in der folgenden Reihenfolge:

- der Schrifttyp wird eingeschaltet,
- die Modifikationen für den Zwischenraum werden eingeschaltet,
- der Positionsbefehl für horizontale Bewegungen wird gegeben,
- die Modifikationen für den Text werden eingeschaltet,
- der Text wird ausgegeben.

Die ersten vier Punkte werden nur dann ausgeführt, wenn sie notwendig sind. Überschreitet der Text die Papierbreite, so zeigen Punkte am Ende der Zeile dies an.

Teil 7: Der Fontspeicher

1. Fonttabellen

Damit die Textverarbeitung Dokumente formatieren kann, muß sie über Breiten und Höhen der einzelnen Schrifttypen (auch "Fonts" genannt) des Druckers, auf dem das Dokument gedruckt wird, Bescheid wissen. Auch bei dem Ausdruck des Dokuments wird diese Information benötigt. Im EUMEL – System stellt der Fontspeicher diese Information den Formatierprogrammen (*lineform* und *pageform*) und dem EUMEL – Drucker zur Verfügung.

Da nun der Drucker Angaben zur Positionierung in seinen Mikroschritten (kleinste Schrittweite in horizontaler oder vertikalter Richtung) benötigt, liefert die Fonttabelle die Breiten – und Höhenangaben in Mikroschritten und eine Umrechnungseinheit von Schritten in Zentimeter oder umgekehrt. So braucht der EUMEL – Drucker bei Positionierungen keine Umrechnung vorzunehmen. Allerdings müssen die Formatierprogramme auch in Mikroschritten des jeweiligen Druckers rechnen. Dadurch werden jedoch Unterschiede durch Rundungsfehler zwischen dem EUMEL – Drucker und den Formatierprogrammen vermieden.

Bei diesem Konzept können Fonts von verschiedenen Druckern nicht in einer Fonttabelle verwaltet werden, denn unterschiedliche Drucker haben meist verschiedene Mikroschritte. Somit muß es für jeden Drucker mindestens eine Fonttabelle geben.

Es gibt aber auch Fälle, in denen Fonts auf einem Drucker nicht mit anderen Fonts des Druckers zusammengedruckt werden können. Solche Fälle liegen z.B. bei Typenraddruckern vor, die immer nur mit einem Typenrad drucken können und dessen Zeichenbreite hardwaremäßig eingestellt werden muß (z.B. beim Olivetti PR320), bei Druckern, die verschiedene Fonts für Längs – und Querformat haben (z.B. beim Agfa P400), oder bei Druckern, deren Fonts geladen werden (z.B. beim HP 2686). Eine **Fonttabelle** enthält also alle die Fonts eines Druckers, **die auf dem Drucker kompatibel sind**. Es kann mehrere Fonttabellen zu einem Drucker geben.

Die verschiedenen Fonttabellen werden von im Multi – User Betrieb von der Task "configurator" verwaltet. Sie enthält alle Fonttabellen, die auf dem Rechner zur Verfügung stehen (im Single – User Betrieb müssen die Fonttabellen in der Task enthalten sein). Mit dem Kommando

```
fonttable ("Name der Fonttabelle")
```

wird in einer Task die gewünschte Fonttabelle eingestellt. Danach stehen die Fonts dieser Tabelle in der Task zur Verfügung. Die Einstellung der Fonttabelle vererbt sich auf die Sohntasks, d.h. wird eine Sohntask begonnen, so ist dort die Fonttabelle des Vaters eingestellt.

Dazu das folgenden Beispiel:

> Für den Agfa – Drucker P400 gibt es die Fonttabellen "agfa" und "agfaquer", in denen die Fonts für Längsdruck bzw. Querdruck enthalten sind. In der Task *PUBLIC* wird mit dem Kommando *fonttable ("agfa")* die Fonttabelle "agfa" eingestellt. Alle neuen Sohntasks können sofort ohne weitere Einstellung mit der Textformatierung im Längsformat beginnen. Will nun jemand im Querformat drucken, so muß er in seiner Task mit dem Kommando *fonttable ("agfaquer")* den Fontspeicher auf die Fonts zum Querdruck umstellen.

Das Kommando

```
list fonts
```

listet die Fonts der eingestellten Fontabelle ins *notebook* und das Kommando

```
list fonttables
```

informiert über die verfügbaren Fonttabellen.

2. Erstellen einer Fonttabelle

Die Fonttabelle ist ein Datenraum mit einer eigenen Struktur. Somit kann sie nicht mehr mit dem Editor, sondern nur mit einem entsprechenden Programm bearbeitet werden. Solch ein Programm befindet sich in der Datei "font convertor" auf dem Standardarchiv "std.175" Diese Datei sollte in einer Systemtask (Sohntask von "SYSUR") insertiert werden. Danach stehen entsprechende Kommandos zur Bearbeitung einer Fonttabelle zur Verfügung.

Um eine Fonttabelle zu bekommen, muß zuerst eine **Fontdatei** (d.h. eine editierbare Datei mit dem unten beschriebenen Aufbau) angelegt werden. Mit dem Kommando

```
create fonttable ("Name der Fontdatei")
```

werden alle in der Fontdatei spezifizierten Fonttabellen erstellt. Sie liegen als benannte Datenräume in der Task vor und können mit dem Kommando *save* von einer Systemtask an die Task "configurator" gesendet werden. Danach sind diese Fonttabellen in allen Task auf dem Rechner verfügbar und können mit dem *fonttable* – Kommando eingestellt werden.

Soll dagegen eine bestehende Fonttabelle geändert werden, so erstellt das Kommando

```
create fontfile ("Name der Fonttabelle", "Name der Fontdatei")
```

aus der angegebenen Fonttabelle eine Fontdatei. Die Fonttabelle muß dazu in der Task als benannter Datenraum vorliegen (d.h. sie muß eventuell mit *fetch* von der Task "configurator" geholt werden). In der so erstellten Fontdatei können die Änderungen mit dem Editor vorgenommen, mit *create fonttable* die geänderte Fonttabelle erstellt und diese wiederum mit *save* an die Task "configurator" gesendet werden. Mit dem *fonttable* – Kommando kann dann in den bestehenden Tasks die geänderte Fonttabelle eingestellt werden. Alle neuen Tasks erhalten automatisch die geänderte Fonttabelle.

Prozedurbeschreibung der Umwandlungs – Kommandos

Nach der Insertierung der Datei "font convertor" stehen die folgenden Kommandos zur Umwandlung einer Fontdatei in eine Fonttabelle oder umgekehrt zur Verfügung.

create fontfile

PROC create fontfile (TEXT CONST fonttable name, fontfile name)

Zweck: Aus Fonttabelle 'fonttable name' wird eine Fontdatei mit dem angegebenen Name erstellt. Die Fonttabelle muß dabei in der eigenen Task als benannter Datenraum vorliegen.

create fonttable

PROC create fonttable (TEXT CONST fontfile name)

Zweck: Es werden alle Fonttabellen erzeugt, die in der Fontdatei 'fontfile name' angegeben sind. Die Fonttabellen liegen dann als benannte Datenräume in der Task vor.

PROC create fonttable

Zweck: Es werden alle Fonttabellen erzeugt, die in der zuletzt bearbeiteten Datei angegeben sind.

3. Aufbau der Fontdatei

In der Fontdatei können drei Strukturen stehen und zwar Kennungen, Identifkationen und Zeichenspezifikationen.[1)]

Kennungen

Formaler Aufbau: <*Kennung*> : Name 1 [, Name 2] [...] ;

Eine Kennung leitet eine Definition ein. Für die Namen der Namensliste gelten die folgenden Konventionen:

- der Name muß als TEXT – Denoter angegeben werden,
- der Name muß ungleich *niltext* sein,
- Leerzeichen sind im Namen nicht signifikant (d.h. "agfa quer" wird zu "agfaquer").

Eine Kennung kann die folgenden Werte annehmen:

<*Kennung*> ∈ { FONTTABLE, FONT }

- **FONTTABLE**
  Hiermit wird eine Definition einer Fonttabelle eingeleitet. Es wird nur der erste Name der Namensliste ausgewertet, da die Fonttabelle eindeutig identifiziert sein muß. Alle folgenden Angaben werden dieser Fonttabelle zugeordnet, bis eine neue Kennung FONTTABLE folgt.

- **FONT**
  Hiermit wird eine Definition eines Schrifttyps eingeleitet. Ein Schrifttyp kann mehrere Namen haben. Jedoch darf in einer Fonttabelle jeder Fontname nur einem Font zugeordnet werden.

1) Beim formalen Aufbau bedeuten eckige Klammern, daß diese Angaben optional sind.

Identifikationen

Formaler Aufbau: [<*Identifikation*> = <Wert der Identifikation> ;]

Mit den Identifikationen werden bestimmte Angaben zu den Kennungen gemacht. Sie müssen unmittelbar nach der entsprechenden Kennung folgen, brauchen aber nur angegeben werden, wenn sie von den Standardwerten abweichen.

Identifikationen nach der Kennung FONTTABLE

<*Identifikation*> ∈ { x unit, y unit, on string, off string }

- **x unit**
  Hiermit wird die Anzahl der Mikroschritte des Druckers pro Zentimeter in horizontaler (x–) Richtung spezifiziert. Die Einheit muß als REAL–Denoter angegeben werden. Alle weiteren Breitenangaben zu den Fonts dieses Druckers beziehen sich auf diese Einheit.

  STD–Wert: 10.0 / 2.54 = 3.937008

- **y unit**
  Hiermit wird die Anzahl der Mikroschritte des Druckers pro Zentimeter in vertikaler (y–) Richtung spezifiziert. Die Einheit muß als REAL–Denoter angegeben werden. Alle weiteren Höhenangaben zu den Fonts dieses Druckers beziehen sich auf diese Einheit.

  STD–Wert: 6.0 / 2.54 = 2.362205

- **on string**
  Hier müssen vier Textdenoter, durch Komma getrennt, angegeben werden. Die Textdenoter enthalten die Befehlssequenzen, um beim Drucker die Modifikationen anzuschalten. Dabei ist die Reihenfolge der Modifikationen
  underline, bold, italics, reverse.
  Liegt für eine der Modifikationen keine Befehlssequenz vor, so muß *niltext* angegeben werden. Die Befehlssequenzen können vom Druckertreiber abgefragt werden.

  STD–Wert: *niltext* für alle Modifikationen

– **off string**
Hier müssen vier Textdenoter, durch Komma getrennt, angegeben werden. Die Textdenoter enthalten die Befehlssequenzen, um beim Drucker die Modifikationen auszuschalten. Dabei ist die Reihenfolge der Modifikationen
underline, bold, italics, reverse.
Liegt für eine der Modifikationen keine Befehlssequenz vor, so muß *niltext* angegeben werden. Die Befehlssequenzen können vom Druckertreiber abgefragt werden.

STD – Wert: *niltext* für alle Modifikationen

Identifikationen nach der Kennung FONT

<Identifikation> ∈ { font lead, font height, font depth, indentation pitch,
next larger font, next smaller font,
font string, y offsets, bold offset }

– **font lead**[1)]
Der Durchschuß eines Fonts gibt den Zwischenraum in vertikaler Richtung zwischen den Zeilen bei einfachem Zeilenvorschub an. Er muß in Mikroschritten der y – Richtung als INT – Denoter angegeben werden.

STD – Wert: 0

– **font height**[1)]
Die Fonthöhe ist die Distanz von der Basislinie bis zur Oberkante des höchsten Zeichens. Sie muß in Mikroschritten der y – Richtung als INT – Denoter angegeben werden.

STD – Wert: 6 Zeilen pro Inch entsprechend der definierten *y unit*

– **font depth**[1)]
Die Fonttiefe ist die Distanz von der Basislinie bis zur Unterkante des tiefsten Zeichens. Sie muß in Mikroschritten der y – Richtung als INT – Denoter angegeben werden.

STD – Wert: 0

1) Für spätere Erweiterungen des EUMEL – Druckers wurde die bisherige Fonthöhe in Durchschuß, Fonthöhe und Fonttiefe aufgespalten. Für alle bis jetzt definierten Leistungen braucht nur wie bisher die Fonthöhe angegeben zu werden. Der Durchschuß und die Fonttiefe werden dann auf Null gesetzt.

– **indentation pitch**
Einrückungen oder Aufzählungen werden äquidistant berechnet, d.h. Anzahl der Zeichen mal einer festen Breite. Diese Einrückbreite sollte ein Mittel aller Zeichenbreiten sein und braucht nicht der Breite des Leerzeichens zu entsprechen. Sie muß in Mikroschritten der x – Richtung als INT – Denoter angegeben werden.

STD – Wert: 10 Zeichen pro Inch entsprechend der definierten *x unit*

– **next larger font**
Hier muß der Name des nächst größeren Fonts als TEXT – Denoter aufgeführt werden. Gibt es keinen nächst größeren Font, so ist *niltext* anzugeben.

STD – Wert: *niltext*

– **next smaller font**
Hier muß der Name des nächst kleineren Fonts als TEXT – Denoter aufgeführt werden. Gibt es keinen nächst kleineren Font, so ist *niltext* anzugeben. Bei Indizes oder Exponenten wird automatisch auf diesen nächst kleineren Font umgeschaltet.

STD – Wert: *niltext*

– **font string**
Hier kann als TEXT – Denoter eine Befehlssequenz angegeben werden, die den Drucker auf diesen Font umschaltet. Diese Befehlssequenz kann vom Druckertreiber abgefragt werden. Dadurch ist es nicht nötig, daß er die Namen der Fonts kennt.

STD – Wert: *niltext*

– **y offsets**
Um bei Matrixdruckern Schriften zu erzeugen, die höher als eine Nadelreihe sind, müssen entsprechende Bitmuster des Textes an verschiedenen y – Positionen ausgegeben werden. Um diese Anforderung durch den EUMEL – Drucker zu unterstützen, kann hier eine Liste von Verschiebungen von der Basislinie angegeben werden, an denen der Text ein weiteres Mal ausgegeben wird. Dabei bedeuten negative Werte eine Verschiebung oberhalb und positive Werte eine Verschiebung unterhalb der Basislinie. Ist der Wert Null, so wird der Text auf der Basislinie ausgegeben. Die Modifikation *underline* wird bei der Ausgabe des Textes nur an der ersten Verschiebung angestellt.

Die Werte für die Verschiebungen müssen in Mikroschritten der y – Richtung als INT – Denoter angegeben und durch Komma getrennt werden.

STD – Wert: 0

– **bold offset**
Falls der Drucker die Modifikation *bold* nicht beherrscht, versucht der EUMEL – Drucker sie durch Doppeldruck zu simulieren. Der 'bold offset' gibt an, ob und wieviel der zweite Durchgang in x – Richtung verschoben werden soll. Dies ergibt insbesondere bei Laserdruckern, die nicht für alle Schrifttypen einen Bold – Typ haben, einen recht guten Fettdruck. Der Wert muß in Mikroschritten der x – Richtung als INT – Denoter angegeben werden.

STD – Wert: 0

Zeichenspezifikationen

Formaler Aufbau: [<Zeichen> [, <Breite des Zeichens>]
[, <Ersatzdarstellung des Zeichens>] ;]

Nachdem die Identifikationen zu einer Kennung angegeben wurden, können Zeichenspezifikationen folgen, d.h. zu einem Zeichen kann die Breite und/oder eine Ersatzdarstellung spezifiziert werden. Dazu muß zuerst das Zeichen selber als TEXT – Denoter angegeben werden.

– **Breite des Zeichens**
Die Zeichenbreite muß als INT – Denoter in Mikroschritten angegeben werden. Alle Zeichenbreiten werden mit der Einrückbreite vorbesetzt, so daß nur solche Zeichen angegeben werden müssen, deren Breite von der Einrückbreite abweichen. Negative Zeichenbreiten sind nicht erlaubt. Die Angabe von Zeichenbreiten nach der Kennung FONTTABLE wird ignoriert.

– **Ersatzdarstellung des Zeichens**
Die Ersatzdarstellung wird statt des Zeichens ausgedruckt. Sie muß als TEXT – Denoter angegeben werden. Werden Ersatzdarstellungen nach der Kennung FONTTABLE angegeben, so gelten sie global für alle Fonts dieser Fonttabelle. Sie können jedoch bei der Fontangabe lokal wieder überschrieben werden. Eine Ersatzdarstellung darf höchsten 255 Zeichen lang sein. Alle Ersatzdarstellungen eines Fonts dürfen 32767 Zeichen nicht überschreiten.

Kommentare in der Fontdatei

In der Fontdatei dürfen Kommentare eingefügt werden. Sie müssen den Kommentaren der ELAN – Syntax entsprechen, d.h. mit '(*' beginnen und mit '*)' enden.

Deutsche Namen

Kennungen und Identifikationen dürfen in der Fontdatei auch mit folgenden deutschen Namen angegeben werden.

FONTTABLE	FONTTABELLE
FONT	FONT
x unit	x einheit
y unit	y einheit
on string	on sequenz
off string	off sequenz
indentation pitch	einrueckbreite
font lead	durchschuss
font height	fonthoehe
font depth	fonttiefe
next larger font	groesserer font
next smaller font	kleinerer font
font string	font sequenz
y offsets	y verschiebungen
bold offset	bold verschiebung

4. Beispiel für eine Fontdatei

In diesem Beispiel einer Fonttdatei sind drei Fonttabellen enthalten, nämlich "agfa" und "agfaquer" für den Agfa – Drucker und "epson" für einen Epson – Drucker.

```
FONTTABLE  :  "agfa" ;
  x unit       =  160.0 ;                          (* Anzahl der Mikroschritte pro cm *)
  y unit       =  160.0 ;
  on string    =  "\UL1;",  "\BO1;",  "\IT1;",  "\CFW;\CBB;" ;
  off string   =  "\UL0;",  "\BO0;",  "\IT0;",  "\CFT;\CBT;" ;

                                  (* globale Ersatzdarstellungen für alle Agfa-Fonts *)

  ""214"" ,   "\!298;" ;                                                  (* AE *)
  ""215"" ,   "\!299;" ;                                                  (* OE *)
  ""216"" ,   "\!300;" ;                                                  (* UE *)
  ""217"" ,   "\!451;" ;                                                  (* ae *)
  ""218"" ,   "\!452;" ;                                                  (* oe *)
  ""219"" ,   "\!453;" ;                                                  (* ue *)



  FONT  :  "trium10" ;
    indentation pitch   =  30 ;
    font lead           =  7 ;
    font heigth         =  54 ;
    font depth          =  15 ;
    next larger font    =  "trium12" ;
    next smaller font   =  "trium8" ;
    font string         =  "\F05;" ;

    " "     ,    20 ;          "!"  ,     16 ;
    """"    ,    22 ;          "#"  ,     31 ;
    "$"     ,    31 ;          "%"  ,     55 ;
    .
    .
    .
    ""217"" ,    31 ;                                                   (* ae          *)

                                  (* lokale Ersatzdarstellungen für Font "trium10" *)

    ""244"" ,    43 ,   "\F023;\!725;\F05;" ;                           (* ungleich    *)
    ""245"" ,    31 ,   "\F023;\!405;\F05;" ;                           (* mal-Zeichen *)
```

```
  FONT  :  "modern12",  "elanlist" ;                    (* Mehrere Namen für einen Font *)
    indentation pitch  =  33 ;
    font lead          =  14;
    font heigth        =  53;
    font depth         =  13;
    next larger font   =  "" ;
    next smaller font  =  "micro" ;
    font string        =  "\FO11;"
                                          (* Alle Zeichen haben die gleiche Breite *)

  FONT . . .


FONTTABLE  :  "agfaquer" ;
  x unit      =  160.0 ;
  y unit      =  160.0 ;
  on string   =  "\UL1;",  "\BO1;",  "\IT1;",  "\CFW;\CBB;" ;
  off string  =  "\UL0;",  "\BO0;",  "\IT0;",  "\CFT;\CBT;" ;
  .
  .
  .


FONTTABLE  :  "epson" ;
  x unit      =  47.24409 ;                                  (* 120.0 / 2.54 *)
  y unit      =  85.03937 ;                                  (* 216.0 / 2.54 *)
  on string   =  ""27"-"1"",  "",  ""27"4",  "";
  off string  =  ""27"-"0"",  "",  ""27"5",  "";

  ""214"" ,   ""27"R"2""091""27"R"0"" ;          (* AE                          *)
  ""215"" ,   ""27"R"2""092""27"R"0"" ;          (* OE                          *)
  ""216"" ,   ""27"R"2""093""27"R"0"" ;          (* UE                          *)
  ""217"" ,   ""27"R"2""123""27"R"0"" ;          (* ae                          *)
  ""218"" ,   ""27"R"2""124""27"R"0"" ;          (* oe                          *)
  ""219"" ,   ""27"R"2""125""27"R"0"" ;          (* ue                          *)
  ""220"" ,   "k" ;                              (* Trenn-k                     *)
  ""221"" ,   "-" ;                              (* Trennstrich                 *)
  ""222"" ,   "#" ;                              (* geschütztes Nummernkreuz    *)
  ""223"" ,   " " ;                              (* geschütztes Leerzeichen     *)
  ""251"" ,   ""27"R"2""126""27"R"0"" ;          (* ss                          *)
  ""252"" ,   ""27"R"2""064""27"R"0"" ;          (* Paragraph                   *)
```

```
FONT  :  "12",  "elite",  "elite12" ;              (* Mehrere Namen für einen Font *)
  font height        =  36 ;
  indentation pitch  =  10 ;
  next smaller font  =  "12.klein" ;
  font string        =  ""27"!"1""27"p"0""27"T" ;
  bold offset        =  2 ;

FONT  :  "12.klein",  "elite.klein",  "elanlist" ;
  font height        =  20 ;
  indentation pitch  =  10 ;
  next smaller font  =  "12.klein" ;
  font string        =  ""27"!"1""27"p"0""27"S"1"" ;
  bold offset        =  1 ;

FONT  :  "12.hoch" ;
  font height        =  96 ;
  indentation pitch  =  10 ;
  next smaller font  =  "12.klein" ;
  font string        =  "" ;
  bold offset        =  2 ;
  y offsets          =  12, -12 ; (* der Text wird jeweils 12 Mikroschritte unter-
                                     und überhalb der Basislinie ausgegeben *)

FONT  :  "prop10",  "prop" ;
  font height        =  12 ;
  indentation pitch  =  24 ;
  next smaller font  =  "" ;
  font string        =  ""27"!"0""27"p"1""27"T" ;
  bold offset        =  2 ;

  "!"   ,   10 ;
  """"  ,   16 ;
  "("   ,   12 ;
  . . .
```

5. Schnittstelle des Fontspeichers

Das Paket *font store* liefert die folgenden Prozeduren:

fonttable

PROC fonttable (TEXT CONST fonttable name)

Zweck: Stellt die angegebene Fonttabelle in der Task ein. Dabei wird zuerst in der eigenen Task nach der angegebenen Fonttabelle gesucht. Existiert die Fonttabelle in der eigenen Task nicht, so wird die Fonttabelle von der Task "configurator" geholt.
Wenn die Fonttabelle eingestellt ist, sind in der Task nur noch die Fonts dieser Fonttabelle bekannt. Die Einstellung vererbt sich auf die Sohntasks.

TEXT PROC fonttable

Zweck: Liefert den Name der eingestellten Fonttabelle.

list fonttables

PROC list fonttables

Zweck: Zeigt die Liste der verfügbaren Fonttabellen im *notebook*.

list fonts

PROC list fonts

Zweck: Listet die Fonts der eingestellten Tabelle ins *notebook*.

PROC list fonts (TEXT CONST fonttable name)

Zweck: Listet die Fonts der angegebenen Fonttabelle ins *notebook*. Die vorher eingestellte Fonttabelle bleibt jedoch weiter eingestellt.

x step conversion

INT PROC x step conversion (REAL CONST cm)

Zweck: Rechnet die in Zentimeter angegebene Länge in Mikroschritte der x – Richtung um.

REAL PROC x step conversion (INT CONST steps)

Zweck: Rechnet die in Mikroschritten der x – Richtung angegebene Länge in Zentimeter um.

y step conversion

INT PROC y step conversion (REAL CONST cm)

Zweck: Rechnet die in Zentimeter angegebene Länge in Mikroschritte der y – Richtung um.

REAL PROC y step conversion (INT CONST steps)

Zweck: Rechnet die in Mikroschritten der y – Richtung angegebene Länge in Zentimeter um.

on string

TEXT PROC on string (INT CONST modification)

Zweck: Liefert die in der Fonttabelle spezifizierte Befehlssequenz, um eine Modifikation anzuschalten. Es gibt die folgenden Modifikationen

1	underline
2	bold
4	italics
8	reverse

off string

TEXT PROC off string (INT CONST modification)

Zweck: Liefert die in der Fonttabelle spezifizierte Befehlssequenz, um eine Modifikation auszuschalten. Es gibt die folgenden Modifikationen

1	underline
2	bold
4	italics
8	reverse

font

INT PROC font (TEXT CONST font name)

Zweck: Liefert die interne Fontnummer des Fonts. Mit dieser Fontnummer können die weiteren Informationen über den Font angefordert werden. Existiert kein Font mit diesem Namen, so wird Null geliefert.

TEXT PROC font (TEXT CONST font nr)

Zweck: Liefert den Fontnamen des Fonts mit der angegeben Fontnummer. Hat der Font mehrere Namen, so wird der erste Name der Namensliste aus der Fontdatei geliefert. Existiert kein Font unter dieser Nummer, so wird *niltext* geliefert.

font exists

BOOL PROC font exists (TEXT CONST font name)

Zweck: Informationsprozedur zur Abfrage der Existenz eines Fonts.

next smaller font exists

BOOL PROC next smaller font exists (INT CONST font nr,
INT VAR next smaller font)

Zweck: Informationsprozedur zur Abfrage der Existenz des nächst kleineren Fonts. Wenn er existiert, wird die Fontnummer dieses Fonts zurückgeliefert.

next larger font exists

BOOL PROC next larger font exists (INT CONST font nr,
INT VAR next larger font)

Zweck: Informationsprozedur zur Abfrage der Existenz des nächst größeren Fonts. Wenn er existiert, wird die Fontnummer dieses Fonts zurückgeliefert.

indentation pitch

INT PROC indentation pitch (INT CONST font nr)

Zweck: Liefert die Einrückbreite in Mikroschritten der x – Richtung. Sie sollte eine mittlere Breite der Zeichen sein, denn mit ihr werden die Einrükkungen und Aufzählungen berechnet.

font lead

INT PROC font lead (INT CONST font nr)

Zweck: Liefert den Durchschuss des Fonts in Mikroschritten der y – Richtung. Der Druchschuß ist der Zwischenraum zwischen den einzelnen Zeilen bei einfachem Zeilenvorschub.

font height

INT PROC font height (INT CONST font nr)

Zweck: Liefert die Höhe des Fonts in Mikroschritten der y – Richtung. Die Fonthöhe ist die Distanz von der Basislinie bis zur Oberkante des höchsten Zeichens.

font depth

INT PROC font depth (INT CONST font nr)

Zweck: Liefert die Tiefe des Fonts in Mikroschritten der y – Richtung. Die Fonttiefe ist die Distanz von der Basislinie bis zur Unterkante des tiefsten Zeichens.

font string

TEXT PROC font string (INT CONST font nr)

Zweck: Liefert den Fontstring des Fonts. Der Fontstring enthält die Befehlssequenz, um den Drucker auf diesen Font umzuschalten.

y offsets

TEXT PROC y offsets (INT CONST font nr)

Zweck: Liefert einen Text mit den y – Verschiebungen von der Basislinie. Die einzelnen Verschiebungen können mit dem Operator 'ISUB' abgefragt werden.

bold offsets

INT PROC bold offsets (INT CONST font nr)

Zweck: Liefert die 'bold' – Verschiebung.

char pitch

INT PROC char pitch (INT CONST font nr, TEXT CONST char)

Zweck: Liefert die Breite des Zeichens in Mikroschritten der x – Richtung.

replacement

TEXT PROC replacement (INT CONST font nr, TEXT CONST char)

Zweck: Falls das Zeichen eine Ersatzdarstellung hat, so wird diese geliefert, anderfalls das Zeichen selbst.

get font

PROC get font (INT CONST font nr,
INT VAR indentation pitch, font lead, font height, font depth,
ROW 256 INT VAR pitch table)

Zweck: Die Variablen liefern die entsprechenden Informantionen über den Font. Der Eintrag des Codewerts eines Zeichens plus eins in der Breitentabelle liefert die Breite dieses Zeichens.

get replacements

PROC get replacements (INT CONST font nr,
TEXT VAR replacements,
ROW 256 INT VAR replacement table)

Zweck: In der Fonttabelle kann für jedes Zeichen eine Ersatzdarstellung angegeben werden. Diese Ersatzdarstellungen werden mit dieser Prozedur geliefert. Dabei stehen in der Textvariablen 'replacement' die gesamten Ersatzdarstellungen des Fonts. Die Ersatzdarstellungstabelle enthält Zeiger auf den Text der Ersatzdarstellungen. Die Ersatzdarstellung eines Zeichnes bestimmt sich wie folgt:

```
ersatzdarstellung :
  INT CONST wert := replacement table (code( zeichen ) + 1);
  IF wert > 0
     THEN INT CONST ende := wert + code (replacements SUB
     wert);
          subtext (replacements, wert + 1, ende)
     ELSE zeichen
   FI.
```

Bei den Prozeduren des Packets *font store* können die folgenden Fehlerfälle auftreten:

- Fonttabelle noch nicht eingestellt
  Es wurde noch keine Fonttabelle in der Task eingestellt.

- Fonttabelle "fonttable name" gibt es nicht
  Die angegebene Fonttabelle wurde weder in der eigenen Task, noch in der Task 'configurator' gefunden.

- Font 'font nr' gibt es nicht
  Unter der angegebenen Fontnummer gibt es in der eingestellten Fonttabelle keinen Font. Speziell ist das für 'font nr' = 0 der Fall, falls ein Fontname nicht gefunden wurde.

- unzulaessige Modifikation
  Die angegebene Modifikation ist ungleich 1, 2, 4 oder 8.

Teil 8: Verschiedenes

1. Installation der Graphik – Pakete

Das Graphik System besteht aus drei Teilen :

1a) geräteunabhängiger Teil des "... plot" – Pakets
1b) geräteabhängiger Teil des "... plot" – Pakets
2) Das geräteunabhängige "plotten" – Paket

Zur Installation des Systems für ein bestimmtes Endgerät wird das entsprechende plot – Paket ausgewählt oder selbst entwickelt (siehe Benutzerhandbuch). Dann werden zuerst

"... plot"
dann "plotten"

insertiert. Dabei empfiehlt es sich, die Zeilennummerngenerierung für Fehlermeldungen der Pakete mit "check off" abzuschalten, um Code zu sparen. Bei Multi – User Systemen kann man mehrere parallele Graphik – Systeme installieren, z. B. eine Graphik – Vater – Task für normale Terminals und eine Plotter – Task.

Standardmäßig wird u.a. das Paket "std plot" ausgeliefert. Es bedient als Graphik – Endgerät ein normales alphanumerisches Terminal. Wegen des sehr groben Rasters (48*79) ist die Darstellungsqualität natürlich sehr schlecht. Insbesondere machen nahe beieinanderliegende Linien die Darstellung sehr unübersichtlich. Es ist auch nur ein realer Stift vorhanden. Trotz dieser Einschränkung kann man mit diesem Paket einfache Bilder (Funktionen, einfache Körper o.ä.) einigermaßen vernünftig auf einem Terminal darstellen, um im Dialog das Programm auszutesten oder die geeignete Darstellung der Graphik auszuwählen.

Alle weiteren vorhandenen "... plot" – Pakete werden ebenfalls mit ausgeliefert. Diese Pakete werden aber in der Regel nicht von der EUMEL Systemgruppe gewartet, sondern von Installationen, die sie für ihre eigenen Geräte entwickelt haben.

Anschluß neuer Graphik – Geräte

Für den Anschluß eines neuen Graphik – Gerätes muß ein entsprechendes (geräteabhängiges) "... plot" – Paket geschrieben werden. Das Interface zum (geräteunabhängigen) "plotten" – Paket ist folgendermaßen definiert :

begin plot

PROC begin plot

Aktion: Das "plotten" – Paket kündigt damit den Anfang einer Zeichnung an. Die Prozedur kann evtl. notwendige Initialisierungen durchführen (z.B. Terminal in den Graphik – Modus umschalten). Bei vielen Geräten wird sie mit leerer Leistung implementiert werden können. Auf keinen Fall sollte der Bildschirm gelöscht bzw. das Papier am Plotter gewechselt werden. Dafür ist 'clear' zuständig.

clear

PROC clear

Aktion: Das Graphik – Gerät muß in die Grundstellung gebracht und eine freie Zeichenfläche zur Verfügung gestellt werden. Das wird beim Graphik – Terminal in der Regel Löschen des Bildschirms und beim Plotter Papierwechsel heißen.

cursor on

PROC cursor on

Aktion: Das Endgerät sollte einen graphischen Cursor einschalten falls dieser vorhanden ist.

cursor off

PROC cursor off

Aktion: Das Endgerät sollte einen graphischen Cursor ausschalten falls dieser vorhanden ist.

dir draw

PROC dir draw (REAL CONST x, y, z)

Aktion: Der aktuelle Stift muß gesenkt zu den Rasterkoordinaten 'h,v', die den Weltkoordinaten 'x, y, z' entsprechen, gefahren werden ("zeichnen"). Die Weltkoordinaten können dabei einen Überlauf produzieren.

PROC dir draw (TEXT CONST text, REAL CONST angle, height)

Aktion: Beginnend an der aktuellen Stiftposition sollte der angegebene Text mit der Buchstabengröße 'height' und dem Winkel 'angle' gegenüber der Waagerechten geschrieben werden. Größe 0.0 bedeutet dabei Standardgröße (geräteabhängig). Es sollte jeweils die beste dem Gerät mögliche Annäherung an Größe und Winkel gewählt werden. Der Stift muß danach wieder auf der Ausgangsposition stehen.

end plot

PROC end plot

Aktion: Das "plotten" – Paket kündigt damit das Ende einer Zeichnung an. Die Prozedur kann evtl. notwendige Abschlußmaßnahmen durchführen (z.B.: Terminal in den Text – Modus umschalten). Bei vielen Geräten wird sie mit leerer Leistung implementiert werden können.

dir move

PROC dir move (REAL COST x,y,z)

Aktion: Der aktuelle Stift muß gehoben zu den Rasterkoordinaten 'h,v', die Weltkoordinaten 'x,y,z' entsprechen, gefahren werden ("zeichnen"). Die Weltkoordinaten können dabei einen Überlauf produzieren.

graphic get

PROC graphic get (TEXT VAR t, REAL VAR x,y)

Aktion: Es sollte die aktuelle Stiftposition in Weltkoordinaten und ein evtl. eingegebener Text zurückgeliefert werden. Bei den meisten Endgeräten wird man nur eine leere Leistung implementieren können.

pen

PROC pen (INT CONST colour, thickness, line type)

Aktion: Es sollte zu dem realen Stift mit der bestmöglichen Annäherung an die gegebenen Werte umgeschaltet werden. Dabei sind die im Benutzerhandbuch definierten Standards für 'colour', 'thickness' und 'line type' zu beachten.

set values

```
PROC set values (ROW 3 ROW 2 REAL CONST size,
                 ROW 2 ROW 2 REAL CONST limits,
                 ROW 3 REAL CONST angles,
                 ROW 2 REAL CONST oblique,
                 ROW 3 REAL CONST perspective)
```

Aktion: Die Werte für die Darstellungsart werden gesetzt und die Transformationsmatrix wird aufgebaut. Dieser Prozedur kann vom "std plot" – Paket übernommen werden, es müssen allerdings die Werte 'display hor' und 'display vert' auf die Rastergröße gesetzt werden.

transform

```
PROC transform (REAL CONST x, y, z, REAL VAR h, v)
```

Aktion: Der dreidimensionale Vektor 'x, y, z' wird in den zweidimensionalen Vektor 'h, v' transformiert. Diese Prozedur kann ebenfalls vom "std plot" – Paket übernommen werden.

2. Der Spoolmanager

Der "Spoolmanager" verwaltet eine Warteschlange von Datenräumen (Dateien), die von einem "Server" abgearbeitet werden sollen. Dabei puffert der Spoolmanager Dateien, die von beliebigen Tasks geschickt werden können, in einer Warteschlange und gibt sie der Reihe nach dem Server zur eigentlichen Verarbeitung. Ein typischer Einsatzfall (aber nicht der einzige) für ein solches System ist der Druck von Dateien in Multi – User – Systemen. Unabhängig davon, ob der Drucker gerade aktiv ist und wieviele Dateien noch auf den Ausdruck warten, kann jeder seine Datei dem Drukkerspool (in der Regel die Task "PRINTER") senden und sofort danach weiterarbeiten.

Prozeduren des Spoolmanagers

Im privilegierten Ast des Taskbaumes (Söhne von "SYSUR"), stehen die folgenden Prozeduren zur Einrichtung eines Spoolmanagers zur Verfügung.

spool manager

PROC spool manager (PROC server, BOOL CONST with start)

Zweck: Die Task, in der die Prozedur aufgerufen wird, wird zum Spoolmanager. Wenn 'with start' auf TRUE gesetzt ist, wird eine Server – Task als unbenannter Sohn (" – ") eingerichtet und mit der übergebenen 'PROC server' gestartet. Anderfalls muß der Spool durch den Benutzer mit Hilfe der Spoolkommandos (siehe dort) gestartet werden.

PROC spool manager (PROC server)

Zweck: Diese Prozedur ruft die Prozedur 'spool manager' mit 'with start' gleich TRUE auf.

Mit Hilfe der folgenden Prozeduren kann der Spool eingestellt werden.

station only

PROC station only (BOOL CONST flag)

Zweck: Wenn flag auf TRUE gesetzt ist, nimmt der Spooler nur Aufträge von Tasks der eigenen Station entgegen.
Voreinstellung: 'station only (FALSE)'.

BOOL PROC station only
Zweck: liefert TRUE, wenn der Spooler nur von der eigenen Station benutzt werden darf.

spool duty

PROC spool duty (TEXT CONST duty)
Zweck: Mit dieser Prozedur kann ein Text im Spooler eingestellt werden, der die Aufgabe des Spoolers beschreibt. Dieser wird beim 'list' gemeldet.

TEXT PROC spool duty
Zweck: Liefert die eingestellte Text – Beschreibung der Aufgabe des Spools.

spool control task

PROC spool control task (TASK CONST task)
Zweck: Diese Prozedur gibt der Task 'task' und ihrer Söhne die Berechtigung Spoolkommandos (z.B. 'stop' oder 'start') an den Spoolmanager zusenden. Dabei muß die Task auf derselben Station wie der Spool sein und in der Task muß die Datei "spool cmd", die sich auf dem Standardarchiv befindet, insertiert werden.
Wird "SUPERVISOR" als Spoolkontrolltask eingestellt, so können alle Tasks der Station, in denen die Datei "spool cmd" insertiert ist, die Spoolkommandos geben.

TASK PROC spool control task
Zweck: Liefert die Taskidentifikation der Spoolkontrolltask.

server channel

PROC server channel (INT CONST channel)
Zweck: Mit Hilfe dieser Prozedur wird im Spoolmanager eine Kanalnummer eingestellt, die der Server mit der Prozedur 'server channel' abfragen kann.
Fehlerfall:
* falsche Kanalangabe
Der angegebene Kanal ist kleiner als 1 oder größerer als 32.

INT PROC serverchannel
Zweck: Liefert die Nummer des Kanals, der im Spool eingestellt ist.

Anmerkung: Soll im nicht – privilegierten Ast des Taskbaums (Söhne von "PUBLIC") ein Spool eingerichtet werden, so muß dort die Datei "spool manager", die sich auf dem Standardarchiv 'std.zusatz' befindet, insertiert werden.

Spoolkommandos

Ein Spool kann zur Verwaltung der Warteschlange wie jede andere Task ans Terminal gekoppelt werden. Danach stehen die folgenden Spoolkommandos zur Verfügung. Diese Kommandos sind keine Prozeduren, sondern werden nur interpretiert. Sie dürfen also nur alleine eingegeben werden. Nach Beendigung der Verwaltungsaufgaben muß der Spool mit dem Kommando 'break' verlassen werden, da sonst keine weiteren Aufträge an den Spool gesendet werden können und auch die Warteschlange nicht weiter abgearbeitet wird.

stop

Zweck: Die Server – Task wird gelöscht und dadurch der Spool deaktiviert. Der Spool empfängt zwar noch weitere Aufträge und sortiert diese in die Warteschlange ein. Die Warteschlange wird aber nicht weiterabgearbeitet. Ein eventuell von der Server – Task belegter Kanal wird freigegeben.
Ist bei einem 'stop' noch ein Auftrag in Bearbeitung, so wird dieser Auftrag abrupt abgebrochen. Es wird jedoch angefragt, ob der Auftrag nochmal neu an die erste Stelle in der Warteschlange eingetragen werden soll.
Ist ein Spool deaktiviert, so wird dies bei einem 'list' angezeigt,

halt

Zweck: Der Spool deaktiviert sich nach Abarbeitung des Auftrags, der gerade bearbeitet wird. Bei einem 'list' wird dies vermerkt.

start

Zweck: Der Spool wird aktiviert, indem eine neue Server – Task begonnen wird. Ist der Spool zuvor nicht gestoppt worden, so wird zuerst ein 'stop' durchgeführt.
Wurde mit der Prozedur 'server channel' kein Kanal eingestellt, so wird die Warnung

WARNUNG : Serverkanal nicht eingestellt

ausgeben. Der Spool wird trotzdem gestartet.

start (kanal nummer)

Zweck: Vor dem Start des Spools wird zuerst mit der Prozedur 'server channel' der angegebene Kanal eingestellt.

first

Zweck: Im Dialog kann ein Auftrag in der Warteschlange auf den ersten Platz vorgezogen werden.

killer

Zweck: Im Dialog werden alle Aufträge der Warteschlange zum Löschen angeboten.

list spool

Zweck: Der aktuelle Zustand des Spools und die Warteschlange werden gelistet.

Ist nun eine Spoolkontrolltask eingestellt worden (siehe 'spool control task'), so muß in ihr die Datei "spool cmd" insertiert werden. Danach stehen die folgenden Prozeduren zur Verfügung.

stop

PROC stop (TASK CONST spool)

Zweck: Dem Spool 'spool' wird ein 'stop' zugestellt, was den Spool deaktiviert. Wird noch ein Auftrag bearbeitet, so wird angefragt, ob dieser neu eingetragen werden soll.

halt

PROC halt (TASK CONST spool)

Zweck: Dem Spool 'spool' wird ein 'halt' zugestellt, d.h der Spool deaktiviert sich nach Beendigung des aktuellen Auftrags.

wait for halt

PROC wait for halt (TASK CONST spool)

Zweck: Dem Spool 'spool' wird ein 'halt' zugestellt. Die Task wartet jedoch auf eine Rückantwort, die ihr der Spool sendet, wenn er sich nach Beendigung des aktuellen Auftrags deaktiviert hat.

Fehlerfall:

* Task "task name" wartet schon auf halt
  Die angegebene Task wartet schon auf eine Rückantwort des Spools 'spool'.

start

PROC start (TASK CONST spool)

Zweck: Dem Spool 'spool' wird ein 'start' zugestellt, wodurch der Spool sich aktiviert. War der Spool zuvor nicht deaktiviert, so wird er zuerst gestoppt.

first

PROC first (TASK CONST spool)

Zweck: Im Dialog kann einer der Aufträge in der Warteschlange des Spools 'spool' auf den ersten Platz vorgezogen werden.

killer

PROC killer (TASK CONST spool)

Zweck: Im Dialog werden die Aufträge der Warteschlange des Spools 'spool' zum Löschen angeboten.

Arbeitsweise des Servers

Der Server wird vom Spoolmanager mit einer Prozedur gestartet, die die Abarbeitung der Warteschlange vornimmt. Dabei muß diese Prozedur zuerst den Datenraum mit dem 'fetch code' (= 11) holen. Danach kann der Server sich noch mit dem 'fetch param code' (= 21) die Dateiparameter (Dateiname, Schreib- und Lesepaßwort, Sendername und Senderstation) abholen und mit der Bearbeitung des Auftrags beginnen.

Beispiel:

```
LET fetch code       = 11,
    param fetch code = 21;
BOUND STRUCT (TEXT name, write pass, read pass, sendername,
              INT  senderstation ) VAR msg;
DATASPACE VAR ds, param ds;
INT VAR reply;

spool manager (PROC server);

PROC server :
     disable stop;
     continue (server channel);
     REP forget (ds); ds := nilspace;
         call (father, fetch code, ds, reply);
         forget (param ds); param ds := nilspace;
         call (father, param fetch code, param ds, reply);
         msg := param ds;
         execute spool;
         IF is error THEN error treatment FI;
    PER;
END PROC server;

PROC execute spool :
     enable stop;
     ...
```

Senden eines Auftrags an den Spool

Jede Task kann jedem Spool durch Aufruf von 'save' eine Datei senden.

Beispiel:

```
save ("datei name", task ("spool name"))
```

Dieses 'save' – Kommando funktioniert zweiphasig. Dabei wird in der ersten Phase dem Spool die Dateiparameter zugesendet. In der zweiten Phase folgt dann der Datenraum selber. Bei Netzübertragung zu einem Spool ist dieses zweiphasige 'save' jedoch nachteilig. Deshalb können Dateien vom Typ 'FILE' auch mit einem einphasigen 'save' unter dem 'file save code' (= 22) an den Spool gesendet werden. Die *'headline'* dieser Dateien muß jedoch dann auf eine bestimmte Art und Weise aufbereitet werden, so daß sie die Dateiparameter enthält. Beim Aufbau der *'headline'* muß eine Information muß mit dem Code 0 beginnen und dem Code 1 enden. Die Dateiparamter müssen dann mit der folgenden Reihenfolge in die *'headline'* eingetragen werden.

- Dateiname
- Schreibepaßwort
- Lesepaßwort
- Name des Senders
- Station des Senders

Beispiel:

```
...
LET file save code = 22;
DATASPACE VAR ds := old (file name);
FILE VAR file := sequential file (input, ds);
INT VAR reply;
headline (file,    ""0"" + file name +
                ""1""0"" + write password +
                ""1""0"" + read password +
                ""1""0"" + name (myself) +
                ""1""0"" + text (station (myself)) + ""1"");
call (spool task, file save code, ds, reply);
...
```

Der Spoolmanager setzt bei Dateien, die mit dem 'file save code' angeliefert werden die *'headline'* wieder auf den Dateinamen.

Den Benutzer stehen neben dem 'save'-Kommando zur Übertragung einer Datei zum Spool noch die folgenden Kommandos zur Verfügung.

```
save (ALL myself, task ("spool name"))
save (SOME myself, task ("spool name"))
```
übertragung aller bzw. einiger Dateien der eigenen Task zum Spool.

```
erase ("datei name", task ("spool name"))
erase (ALL task ("spool name"), task ("spool name"))
```
Löschen eines bzw. aller eigenen Aufträge in der Warteschlange des Spools

```
list (task ("spool name"))
```
Liste des Spools über den aktuellen Zustand und die Warteschlange.

Existiert ein Spool "PRINTER", so gibt es noch die folgenden Befehle.

```
print
print ("datei name")
print (ALL myself)
print (SOME myself)
```
Sie entsprechen einem 'save' an die Task "PRINTER"

```
printer
```
Liefert den internen Taskbezeichner der Task "PRINTER", d.h. diese Prozedur entspricht dem Aufruf von 'task ("PRINTER")'.

3. Freie Kanäle

Das Paket 'free channel' ermöglicht in Multi – User – Systemen die Einrichtung freier Kanäle. Freie Kanäle kann man zusätzlich zu dem Terminalkanal, der einem vom Supervisor zugeordnet wurde, benutzen. Jeder freie Kanal wird durch eine (benannte) Task – dem Kanalmanager – implementiert. Er wird danach mit dem Tasknamen angesprochen und kann von jeder Task belegt und wieder freigegeben werden. Während einer Belegung können andere Tasks den Kanal nicht benutzen. Der Kanalmanager koppelt sich für jede Belegung an den physikalischen Kanal an und gibt ihn danach auch wieder frei. Ein physischer Kanal kann also im Wechsel von mehreren Kanalmanagern oder einem Kanalmanager und "normalen" Tasks belegt werden.

Das Paket 'free channel' muß beim Kanalmanager und allen Benutzern des Kanals bzw. bei einem gemeinsamen Vater insertiert sein.

FCHANNEL

Zweck: Der Datentyp FCHANNEL spezifiziert einen freien Kanal. Die Assoziierung mit einem realen freien Kanal erfolgt mit der Prozedur 'free channel' und der Zuweisung ':=' (ähnlich wie beim Datentyp FILE).

:=

OP := (FCHANNEL VAR dest, FCHANNEL CONST source)

Zweck: Zuweisung. Wird insbesondere bei der Assoziation (Assoziation: Verbindung zwischen FCHANNEL VAR und Kanal) benötigt.

close

PROC close (FCHANNEL VAR f)

Zweck: Der belegte FCHANNEL wird freigeben.

PROC close (TEXT CONST channel name)

Zweck: Der namentlich spezifizierte Kanal wird freigegeben.

dialogue

PROC dialogue (FCHANNEL CONST f, TEXT CONST end of dialogue char)

Zweck: Der Terminalkanal wird direkt mit dem angegebenen freien Kanal gekoppelt. (Das Benutzerterminal wird "durchgeschaltet".) Eingaben am Terminal werden auf 'f' ausgegeben, auf 'f' ankommende Daten werden auf dem Benutzerterminal ausgegeben. Der Datenverkehr erfolgt im Vollduplexmodus, d.h. der Datenverkehr beider Richtungen läuft unabhängig voneinander parallel. Hiermit können Terminals dynamisch an andere Rechner gekoppelt werden. Der Dialogzustand wird durch Eingabe des 'end of dialogue char' am Benutzerterminal beendet.

fetch

PROC fetch (FCHANNEL VAR channel, TEXT CONST filename, controlchars)

Zweck: Die angegebene datei wird über den Kanal 'channel' eingelesen. Dabei besteht 'control chars' aus zwei bis vier Zeichen

(eof + eol + handshake + handshake prompt)

eof:

Dieses Zeichen wird als Dateiabschluß erwartet.

eol:

Dieses Zeichen wird als Zeilenende erwartet.

handshake, handshake prompt:

Falls 'handshake prompt <> "" ' ist, wird bei dem Empfang eines Prompt-Zeichen eine Quittung (Handshake-Zeichen) ausgegeben.

free channel

FCHANNEL PROC free channel (TEXT CONST channel name)

Zweck: Der namentlich spezifizierte Kanal wird belegt und als FCHANNEL geliefert.

Fehlerfälle:

* task not existing
* channel not free

PROC free channel (INT CONST physical channel number)

Zweck: Installiert die eigene Task als Kanalmanager für den angegebenen physikalischen Kanal.

in

PROC in (FCHANNEL CONST f, TEXT VAR response)

Zweck: Es werden die Daten geliefert, die seit dem letzten 'in'-Aufruf bzw. seit der Assoziierung eintrafen. Bei 'niltext' liegen keine Eingabedaten vor.

PROC open (FCHANNEL VAR f)

Zweck: Der Kanal wird neu belegt. Die Assoziation erfolgt mit dem gleichen Kanal wie bei der letzten Assoziation.

Fehlerfälle:

* "task" gibt es nicht
* Kanal ist nicht frei

out

PROC out (FCHANNEL VAR f, TEXT CONST message)

Zweck: Der übergebene Text wird auf dem Kanal 'f' ausgegeben.

save

PROC save (FCHANNEL VAR f, TEXT CONST name, control chars)

Zweck: Die übergebene Datei muß eine Textdatei sein (Struktur eines FILEs haben). Sie wird komplett auf dem Kanal 'f' ausgegeben.
Dabei bestehen 'control chars' aus bis zu drei Zeichen:
(eof char + eol char + handshake option)

eof char:
Dieses Zeichen wird als Dateiabschluß geschickt.

eol char:
Dieses zeichen wird als Zeilenabschluß geschickt.

handshake option:
Falls die 'control chars' drei Zeichen umfassen, wird nach jeder Zeile auf das als drittes definierte Handshake-Zeichen gewartet.

Beispiele:

```
a) FCHANNEL VAR f := free channel ("otto") ;
   TEXT VAR antwort ;
   out (f, "hallo") ;
   in (f, antwort) ;
   put (antwort) ;
   close (f) ;

b) open (f) ;
   REP
     out (f, "hallo ") ;
     in (f, antwort)
   UNTIL antwort <> "" PER ;
   put (antwort) ;
   close (f) ;

c) open (f) ;
   dialogue (f, "@") ;
   close (f)
```

Stichwortverzeichnis

Anhang zum

Standardarchiv "std.printer"

Stand: 26.2.87

Herausgeber:

Gesellschaft für Mathematik und Datenverarbeitung mbH

Postfach 1240, Schloß Birlinghoven
D – 5205 Sankt Augustin 1
Telefon(02241) 14 – 1, Telex 8 89 469 gmd d
Telefax(02241) 14 28 89, BTX *43900#
Teletex 2627 – 224135 = GMDVV

Autor:

Rudolf Ruland

Texterstellung:

Dieser Text wurde mit der EUMEL – Textverarbeitung erstellt und aufbereitet und mit dem Agfa Laserdrucksystem P400 gedruckt.

Umschlaggestaltung:

Hannelotte Wecken

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.

Anhang zum Standardarchiv "std.printer"

Stand: 26.2.87

Das Standardarchiv "std.printer" enthält einige Anpassungen für diverse Drucker. Diese Anpassungen sind zwar getestet worden, jedoch können noch Fehler auftreten, weil zum einem bei hardwaremäßigen Modifikationen des Herstellers in der Ansteuerung des Druckers die Typbezeichnung des Druckers oft unverändert bleibt und zum anderen bei softwaremäßigen Änderungen nicht immer alle Drucker zu einem Test zur Verfügung standen. So sollten die Druckeranpassungen als Anregung dienen, bei denen eventuell auftretende Fehler beseitigt oder individuelle Wünsche selbst programmiert werden müssen. Die folgende Tabelle zeigt eine Übersicht der augenblicklich vorhandenen Anpassungen.

Name der Druckeranpassung		Hersteller	Druckertypen
"printer.std"		Standardprinter	
"printer.apple"	(M)	APPLE	: Imagewriter
"printer.binder.1550"	(M)	BINDER	: 8510A | 1550
"printer.binder.f10-55"	(T)	BINDER	: F10-55
"printer.canon.lbp-8"	(L)	CANON	: LaserBeamPrinter-8 A1 & A2
"printer.epson.fx"	(M)	EPSON	: FX-80 | FX-100 | FX-100+
"printer.epson.lq"	(M)	EPSON	: LQ-1500
"printer.epson.lx"	(M)	EPSON	: LX-86
"printer.epson.mx"	(M)	EPSON	: MX-80 TYPE III
"printer.epson.rx"	(M)	EPSON	: RX-80 F/T+ | RX-80 F/T
"printer.epson.sq"	(M)	EPSON	: SQ-2500
"printer.fujitsu.dpmg9.i"	(M)	FUJITSU	: DPMG9 TYP I
"printer.hp.laserjet"	(L)	HP	: LaserJet | LaserJet+
"printer.hp.thinkjet"	(M)	HP	: ThinkJet
"printer.kyocera.F-1010"	(L)	KYOCERA	: F-1010
"printer.nec.p3-2"	(M)	NEC	: P2/P3-2 | P2/P3-7
"printer.nec.p3-3"	(M)	NEC	: P2/P3-3 | P2/P3-6
"printer.nec.p5"	(M)	NEC	: P5 | P6 | P7

(M): Matrixdrucker; (T): Typraddrucker; (L): Laserdrucker

Name der Druckeranpassung		Hersteller	Druckertypen
"printer.olivetti.et221"	(T)	OLIVETTI	: ET201 | ET221
"printer.olivetti.pr1470"	(M)	OLIVETTI	: PR1470
"printer.olivetti.pr1450"	(M)	OLIVETTI	: PR1450
"printer.olivetti.pr17b"	(M)	OLIVETTI	: PR15B | PR17B
"printer.olivetti.pr320"	(T)	OLIVETTI	: PR320
"printer.siemens.pt-11"	(M)	SIEMENS	: PT88-11 | PT89-11
"printer.siemens.pt-15"	(M)	SIEMENS	: PT88-15 | PT89-15
"printer.star.nl"	(M)	STAR	: NL-10
"printer.star.sg"	(M)	STAR	: SG-10 | SG-15
"printer.uchida.dwx-305"	(T)	UCHIDA	: DWX-305

(M): Matrixdrucker; (T): Typraddrucker; (L): Laserdrucker

Die Druckeranpassung *"printer.std"* ist eine universelle Druckeranpassung für alle Drucker, die mit ASCII-Code 13 ein 'Carriage Return' (d.h. Bewegung des Druckkopfes an den linken Rand) und mit ASCII-Code 10 eine Zeilenschaltung von 1/6 Zoll vornehmen. Mit ihr kann dann in einem Schrifttyp (entweder 10 oder 12 Zeichen pro Zoll, je nachdem welche Fonttabelle eingestellt ist) gedruckt werden.

Die Namen der Fonttabellen zu den Druckeranpassungen beginnen mit "fonttab.", gefolgt von der Typbezeichnung. Um nun eine dieser Anpassungen zu benutzen, müssen die folgenden Schritte vollzogen werden (der ersten Schritt ist nur im Multi-User Betrieb notwendig).

- Einrichten der Task "PRINTER" als Sohntask von "SYSUR" mit dem Supervisor-Kommando:
  ```
  begin ("PRINTER", "SYSUR")
  ```

- Anmelden des Archivs:
  ```
  archive ("std.printer")
  ```

- Holen der Druckeranpassung vom Archiv:
  ```
  fetch ("printer.druckertyp", archive)
  ```

- Ausschalten der Zeilennummergenerierung bei der Insertierung:
  ```
  check off
  ```

- Insertieren der Druckeranpassung:
  ```
  insert ("printer.druckertyp")
  ```

Nach der Insertierung wird zuerst nach dem Druckerkanal gefragt. Dann werden druckerspezifische Fragen zur Papierbreite, Positionierungsart oder ähnlichem gestellt, die mit 'j' oder 'n' beanwortet werden müssen. Dabei werden alle Alternativanworten zu der jeweiligen Frage hintereinander angeboten, bis eine der Alternativen mit 'j' beanwortet wird. Schließlich muß nochmals das Archiv mit der entsprechenden Fonttabelle eingelegt werden, um diese zu laden. Wenn die Generierung beendet ist, muß im Multi-User Betrieb in allen bestehenden Tasks – insbesondere in der Task "PUBLIC" – die Fonttabelle mit dem Kommando

```
fonttable ("fonttab.druckertyp")
```

eingestellt werden. Danach kann man sich mit

```
list fonts
```

über die vorhandenen Schrifttypen informieren und mit

```
print ("dateiname")
```

eine Datei ausgedrucken.

Der Druckerkanal muß mit der Konfigurationstabelle **'transparent'** konfiguriert werden.

Einstellmöglichkeiten der Druckeranpassungen

a) Prozeduren

Bei den ausgelieferten Druckeranpassungen lassen sich mit den folgenden Prozeduren Einstellungen vornehmen. Die Einstellungen können teilweise mit der *material*-Anweisung oder durch direkte Druckeranweisungen vom Benutzer verändert werden. Bei einer Änderung einer Einstellung muß die entsprechende Prozedur in der Task "PRINTER" mit den neuen Werten aufgerufen und der Spool mit dem Spoolkommando 'start' neu gestartet werden.

* papersize

Druckeranpassungen: alle

PROC papersize (REAL CONST x size, y size)
Zweck: Hiermit wird der Druckeranpassung mitgeteilt, welche Werte für die Papiergröße dem EUMEL-Drucker bei der Funktion 'open document'

geliefert werden sollen. Der EUMEL – Drucker bedruckt nur diese Fläche. Die Konstanten 'x size' und 'y size' geben die Größe dieser Fläche in Zentimetern an.

PROC papersize
Zweck: Informationsprozedur über die eingestellten Werte.

* std speed

Druckeranpassungen: "printer.epson.fx", "printer.epson.lx",
"printer.epson.rx", "printer.epson.mx",
"printer.fujitsu.dpmg9.i", "printer.hp.thinkjet",
"printer.olivetti.17b", "printer.olivetti.et221",
"printer.siemens.pt – 11", "printer.siemens.pt – 15",
"printer.star – nl", "printer.star – sg"

PROC std speed (TEXT CONST speed)
Zweck: Um bei Matrixdruckern horizontal in Mikroschritten positionieren zu können, muß meistens in den Graphikmodus gewechselt werden. Bei den obengenannten Druckern wird der Druck dadurch verlangsamt. Mit dieser Prozedur kann die Positionierungsart eingestellt werden. Wenn die Textkonstante 'speed' gleich "fast" ist, wird in ganzen Blanks des jeweiligen Schrifttyps positioniert. Diese Positionierungsart ist jedoch ungenauer und es sollten verschiedene Schrifttypen bei dieser Positionierungsart nicht miteinander gemischt werden. Ist die Textkonstante 'speed' gleich "slow", so wird auf Positionierung in Mikroschritten (also im Graphikmodus) geschaltet.
Unabhängig von dieser standardmässigen Einstellung kann jeder Benutzer in seiner Datei mit Hilfe der 'material' – Anweisung vor der ersten Zeile die gewünchten Geschwindigkeit und somit Genauigkeit einstellen.
Also wird mit

```
#material ("fast")#
```

beim Druck des Textes nur in ganzen Blanks und mit

```
#material ("slow")#
```

in Mikroschritten positioniert. Fehlt die 'material' – Anweisung so wird die eingestellte Standardgeschwindigkeit genommen.

TEXT PROC std speed
Zweck: Informationsprozedur über den eingestellten Wert.

* std quality

Druckeranpassungen: "printer.epson.lq", "printer.epson.sq",
"printer.nec.p5", "star.nl"

PROC std quality (TEXT CONST quality)

Zweck: Die obengenannten Matrixdrucker haben eine Schönschrift, die genauso breit ist wie die Normalschrift. Mit Hilfe dieser Prozedur kann die standardmäßige Druckqualität eingestellt werden. Ist die Textkonstante 'quality' gleich "nlq" (near letter quality), so werden alle Texte in Schönschrift ausgedruckt. Ist dagegen die Textkonstante "draft", so werden die Text in Entwurfsqualität gedruckt.
Unabhängig von dieser standardmässigen Einstellung kann jeder Benutzer in seiner Datei mit Hilfe der 'material' – Anweisung vor der ersten Zeile die gewünchte Druckqualität einstellen.
Also wird mit

```
#material ("nlq")#
```

der Text in Schönschrift und mit

```
#material ("draft")#
```

in Normalschrift gedruckt. Fehlt die 'material' – Anweisung so wird die eingestellte Standardqualität genommen.
Innerhalb des Textes kann durch die direkten Druckeranweisungen

```
#"nlq"#  oder  #"draft"#
```

auf Schönschrift oder Normalschrift geschaltet werden.

TEXT PROC std quality

Zweck: Informationsprozedur über den eingestellten Wert.

* std typeface

Druckeranpassungen: "printer.epson.sq"

PROC std typeface (TEXT CONST typeface)

Zweck: Der obige Drucker hat im Schönschriftmodus verschiedene Schriftarten. Mit Hilfe dieser Prozedur wird die standardmäßige Schriftart eingestellt. Zulässige Werte sind

```
"roman"
"sansserif"
"courier"
"prestige"
"script"
```

Diese standardmäßige Schriftart kann wieder mit Hilfe der Materialanweisung für ein Dokument und durch direkte Druckeranweisungen innerhalb des Dokuments geändert werden.

TEXT PROC std typeface
Zweck: Informationsprozedur über den eingestellten Wert.

* paper feed

Druckeranpassungen: "printer.binder.f10 – 55", "printer.epson.rx", "printer.nec.p3 – 2", "printer.nec.p5", "printer.olivetti.pr320"

PROC paper feed (TEXT CONST name)
Zweck: Mit dieser Prozedur wird der Druckeranpassung mitgeteilt, ob das Papier mit automatischem Einzelblatteinzug ('name' gleich "sheet") oder als Endlospapier über Traktor oder Walze ('name' gleich "tractor") eingeführt wird. Bei Einzelblatteinzug muß meist bei der Funktion 'open page' eine andere 'y start' – Position zurückgemeldet werden.
Bei den NEC – Druckern sollte bei Einblatteinzug der Druckertyp mit angegeben werden (z.B. "sheet/p2" oder "sheet/p7"), denn die Druckeranpassung nimmt bei den breiten Druckern eine Zentrierung des Papiers vor.
Für den BINDER F10 – 55 Drucker sind zwei unterschiedliche Einzelblatteinzüge angpaßt, nämlich ASF160/560 und ASF176/576. Deßhalb muß hierbei der Name des gewählten Einzahlblattzuges angegeben werden ('name' gleich "asf160" oder "asf576").

TEXT PROC paper feed
Zweck: Informationsprozedur über den eingestellten Wert.

* asf length

Druckeranpassungen: "printer.binder.f10 – 55"

PROC asf length (INT CONST wert)
Zweck: Beim automatischen Einzelblatteinzug ASF160/560 des BINDER F10 – 55 muß eine Formularlänge eingestellt werden. Der Wert dieser Einstellung muß mit dieser Prozedur dem Druckertreiber bekannt gegeben werden. Bei DINA4 – Papier sollte 13 eingestellt werden.

INT PROC asf length
Zweck: Informationsprozedur über den eingestellten Wert.

* printer type

Druckeranpassungen: "printer.hp.laserjet"

PROC printer type (TEXT CONST name)
Zweck: Hiermit wird dem Druckertreiber mit geteilt, ob es sich bei dem angeschlossenen Drucker um eine HP LaserJet oder um eine HP Laser Jet+ handelt. Ist 'name' gleich "LaserJet+", so stellt sich die Drukkeranpassung auf diesen Typ ein.

TEXT PROC printer type
Zweck: Informationsprozedur über den eingestellten Wert.

b) weitere Materialwerte

Mit Hilfe der *material*-Anweisungen kann beliebige Information an die Druckeranpassung geliefert werden. Die Druckeranpassung kann dann diese Information nach ihren Möglichkeiten – meist zu Beginn eines Dokuments – auswerten. Mehrere Matieralwerte müssen durch Schrägstrich getrennt in einer Materialanweisung angegeben werden. Neben den schon oben erwähnten Materialwerten gibt es noch folgende.

* `#material ("quer")#` oder `#material ("landscape")#`

Druckeranpassungen: "printer.hp.laserjet", "printer.kyocera.f-1010", "printer.nec.p3-2"

Zweck: Die Hardwareanpassung stellt sich auf Querdruck ein und liefert bei 'open document' die durch 'paper size' eingestellten Werte der Papiermaße vertauscht dem EUMEL-Drucker.
Bei den beiden Laserdruckern sollten nur die Schrifttypen angesprochen werden, die auch im 'landscape'-Modus vorhanden sind. Bei dem Matrixdrucker ist dieser Materialwert nur sinnvoll, wenn das Papier mit automatischen Einzelblatteinzug quer eingezogen wird.
Dieser Materialwert wird nur einmal pro Dokument ausgewertet, und auch nur dann, wenn er vor der ersten Textzeile steht.

* `#material ("manual")#` und `#material ("tray")#`

Druckeranpassungen: "printer.hp.laserjet", "printer.kyocera.f – 1010"

Zweck: Nach dem Materialwert "manual" zieht die Hardwareanpassung die nächste Papierseite aus dem Einzelblatteinzug. Diese Einstellung bleibt solange bestehen, bis wieder mit dem Materialwert "tray" die Papierkassette wieder angesteuert wird.

* `#material ("manual")#` und `#material ("sheet1")#` und `#material ("sheet2")#`

Druckeranpassungen: "printer.binder.f10 – 55"

Zweck: Wird der BINDER F10 – 55 mit dem automatischen Einzelblatteinzug ASF176/576 betrieben (d.h. 'paper feed ("asf576")'), dann zieht die Hardwareanpassung nach dem Materialwert "manual" die nächste Papierseite aus dem Einzelblatteinzug, nach "sheet1" aus dem ersten und nach "sheet2" aus dem zweiten Papierschacht. Diese Einstellung bleibt solange bestehen, bis sie durch eine andere abgelöst wird.