Extendable multi User Microprocessor ELAN-system

Systemhandbuch

Version 1.7

Universität Bielefeld
Hochschulrechenzentrum

EUMEL

Extendable multi User Microprocessor ELAN-system

Version 1.7

Systemhandbuch

Stand: 16.5.84

Hochschul-
Rechen-
Zentrum

Universität Bielefeld

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 einschalten" 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

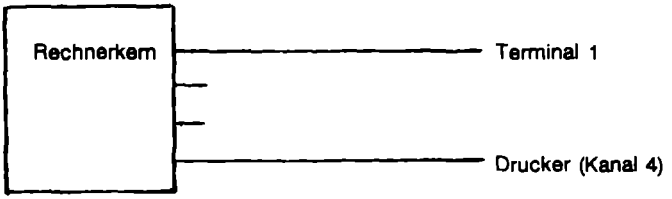
Wie Ihr System aufgebaut ist

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

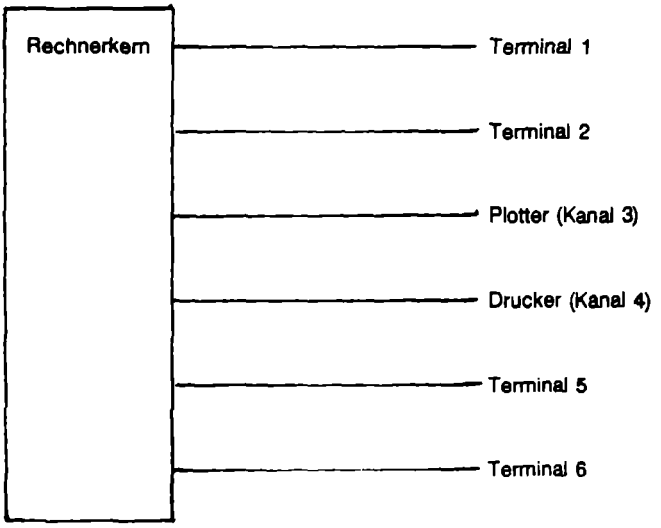
Anmerkung: In manchen Fällen ist das Terminal hardwaremäßig in den Rechner integriert. Trotzdem fassen wir diese physische Einheit dann als zwei logisch getrennte Komponenten auf, nämlich Rechnerkern und Terminal!

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 15 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 eine Diskette mit der Aufschrift "HG" (Hintergrund), eine Diskette mit der Aufschrift "ARCHIV: conf" und mehrere Disketten mit der Aufschrift "ARCHIV: std" (Standardarchive) bekommen haben. Dabei befinden sich auf der Hintergrund-Diskette 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:

> Beim Systemstart führt das EUMEL-System zuerst einen Speichertestdurch. Dabei erscheint das Wort "Speichertest:" auf dem Bildschirm des Hauptterminals. Während des Tests werden "*"-Zeichen ausgegeben. In dieser Zeit drücken Sie die Taste ‹ESC›. Dann wird nicht "durchgestartet", sondern es erscheint ein Startmenü, das Ihnen unter anderem
>
> (1) Systemstart
> (2) Hintergrund von Archiv laden
>
> anbietet. Legen Sie jetzt die Diskette "HG" mit dem Hintergrund ***in das Archivlaufwerk*** ein. Dann (Eingabe von Ihnen sind mit einem Pfeil nach links gekennzeichnet, Ausgaben vom Rechner mit - - - - ›):

‹ - - - - 2

- - - - › 1. HG – Archiv eingelegt (j/n)?

‹ - - - - j

Jetzt wird der Hintergrund eingespielt. Danach erscheint wieder das Startmenü. Wählen Sie jetzt Alternative (1), um das System zu starten.

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 Typen werden auf dem Bildschirm aufgelistet. Sie stellen die Nummer des Kanals und seinen Gerätetyp ein. Beispiel (mit reduzierter Anzahl von Gerätetypen):

- - - - ›	FT10	WS581	F10 – 55	PR1471
	FT10ascii	M20	F10 – 55	PR320
	VC404	ELBIT	FX80	STDPRT
	VC404ascii	ELBITascii	P400	STDPRTascii

- - - - › Kanal:

‹ - - - - 1 ‹ RETURN ›

- - - - › Typ:

‹ - - - - FT10 ‹ RETURN ›

- - - - › Noch ein Kanal zu konfigurieren (j/n)

‹ - - - - j

- - - - › Kanal:

‹ - - - - 4 ‹ RETURN ›

- - - - › Typ:

‹ - - - - P400 ‹ RETURN ›

- - - - › Noch ein Kanal zu konfigurieren (j/n)?

‹ - - - - n

- - - - › Koennen unbenutzte Geraetetypen geloescht werden (j/n)?

‹ - - - - j

Dabei ist folgendes zu beachten:

- In dem Menü sind sowohl Terminal- als auch Druckertypen aufgeführt.
- 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 die Bezeichnung "ascii", 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.

- Wenn die Bezeichnung Ihrer Geräte in dem Menü nicht erscheint, fragen Sie den Lieferanten Ihrer Hard – und Software. Möglicherweise können Sie das Gerät doch ohne weitere Umstände konfigurieren, weil es genauso wie eines der aufgeführten Geräte behandelt werden muß. (Viele verschiedene Typen können gleichartig angeschlossen werden; das Menü ist aber begrenzt.)

- Sie können einen Kanal beliebig oft hintereinander umkonfigurieren. Wenn Sie also einen Fehler gemacht haben, geben Sie einfach den Kanal noch einmal an.

- Wenn Sie alle die Kanäle, die Sie benötigen, richtig konfiguriert haben, antworten Sie auf die Frage "Noch ein Kanal zu konfigurieren (j/n)?" mit "n".

- 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.)

Danach ist Ihr System konfiguriert und Sie können wie in der Beispielsitzung aus dem Kapitel "Einführung" des Benutzerhandbuchs fortfahren. In jedem Fall müssen Sie das System vermittels der SV – Taste anrufen, um ein Supervisor – Kommando geben zu können. Dazu drücken Sie die SV – Taste.

Nicht weiterlesen!

Erst, wenn Sie etwas Erfahrung gesammelt haben, sollten Sie das folgende lesen!

2. Ausführliche Beschreibung

System laden

Wie in der Einführung dieses Kapitels beschrieben ist, geht man beim Systemstart durch Eingabe von ‹ ESC › während des Vortests in das Startmenü und wählt dort "Hintergrund von 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.

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 SYS – Zweiges ebenfalls ein ***"begin" – Paßwort*** definieren. Das geschieht mit dem Kommando

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

Bei der Wahl der Paßworte sollte man folgendes bedenken:

- Ein Paßwort merkt sich leichter als viele verschiedene.
- 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.

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" angekoppeln. (Dabei wird das Paßwort überprüft, falls die Task – wie in der Einführung beschrieben – geschützt wurde.)

Anmerkung: Man kann die Task "configurator" löschen und dann neu (als Sohn, Enkel,... von SYSUR) wieder einrichten. Danach gibt man 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, bleiben 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 "conf" 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.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".

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 "configurate". Den Dialog kann man aber sofort abbrechen, indem man als ersten Kanal "0" angibt.

Konfiguration im Single – User – System

In Single – User – Systemen kann man die Konfiguration nur zu Beginn einmal einstellen. Will man sie ändern, sollte man einen frischen Hintergrund erzeugen und neu konfigurieren. Die Konfiguration kann auch nicht gesichert werden. (Normalerweise dürften auch nur ein Terminal – und ein Druckerkanal von Interesse sei.)

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) explizit durch das Kommando

```
configurate
```

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 "configurate" geben.

Drucker – und Textsoftware im Single – User

In Single – User – Systemen ist (aus Platzgründen) noch keine Druckersoftware insertiert. Ebenfalls sind 'pageform', 'lineform' und 'indexer' nicht generiert. Die entsprechenden Pakete sind deshalb in den Standardarchiven enthalten.

Zur Generierung des Standarddruckers meldet man das Archiv 'std' an:

```
archive ("std")
```

Falls das Standardarchiv auf mehrere Disketten aufgeteilt ist, muß man nun diejenige suchen, die die Datei "std printer generator/S" enthält. Dazu kann man das Kommando

```
list (archive)
```

verwenden. Wenn die richtige Diskette im Laufwerk liegt, kann die Generierung dann erfolgen:

```
fetch ("std printer generator/S", archive)

run
```

Dabei wird die Nummer des Kanals, an den der Drucker angeschlossen ist, im Dialog erfragt. Danach können Dateien mit dem Kommando "print" gedruckt werden.

Anmerkung: Der eigentliche Druckertyp wurde schon bei der Konfigurierung eingestellt.

Wenn die Textsoftware ebenfalls generiert werden soll, geschieht das entsprechend dem obigen Modell. Als Generator-Datei wird allerdings "texter generator/S" verwandt:

```
fetch ("texter generator/S", archive)

run
```

Drucker – und Textsoftware im Multi – User

In Multi – User – Systemen sind sowohl Drucker als auch Textsoftware schon standardmäßig vorhanden. Allerdings muß hier dem Drucker noch mitgeteilt weden, welchen Kanal er benutzen soll. Dazu holt man sich die Task "PRINTER" mit dem Supervisor – Kommando

```
continue ("PRINTER")
```

an ein Terminal und definiert dann den Druckerkanal (hier Kanal 4):

```
start (4)
```

Danach koppelt die Task sich automatisch vom Terminal ab und der Drucker steht zur Verfügung.

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 (V24)
- 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 Kombinierungen 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. Nach Information über generierte Terminals, Speicher und Hintergrund 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 Standardtest 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 ineinander verschränkte 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 fälschlicherweise gekreuzt bzw. nicht gekreuzt, Kabel ohne Flußkontrolle an Schnittstelle mit Flußkontrolle, V24-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).

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 Versuch 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 ***Schreib-/Lesetest*** wird zuerst eine Prüfsumme über den ganzen Hintergrund gebildet. In jedem weiteren Durchlauf wird der Hintergrund blockweise gelesen und wieder zurückgeschrieben. Dabei wird wiederum eine Prüfsumme gebildet und mit der alten verglichen. Der Test bricht bei falscher Prüfsumme oder nach ‹ ESC › an Terminal 1 ab.

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

Der ***Positionierungstest*** 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

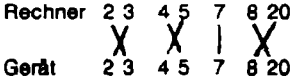
Anmerkung: Dieses Kapitel ist nur für solche Anwender von Bedeutung, die sich selbst mit der Verkabelung ihrer Geräte befassen.

Im folgenden werden die wichtigsten Leitungen der offiziellen V24 – Schnittstelle(serielle Schnittstelle zum Anschluß von Terminals, Druckern, Fremdrechnern u.ä.) beschrieben:

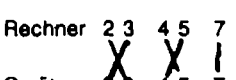
Pin	Betriebsrichtung	Bedeutung
2	out	Sendedaten
3	in	Empfangsdaten
4	out	Sendeaufforderung (RTS)
5	in	Empfangsbereitschft (CTS)
7		Signalerde
8	in	Gegenstation bereit (DCD)
20	out	eigene Station bereit (DTR)

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

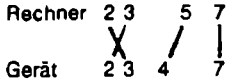
Die meisten Rechner haben die gleiche Pinbelegung wie oben aufgeführt. Die Kabel müssen dann die folgende Pins verbinden:



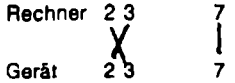
Vollständige Verbindung mit Flußkontrolle.



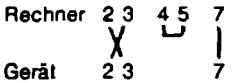
Reicht für die meisten Anschlüsse mit Flußkontrolle, z.B. Rechnerkopplung.



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



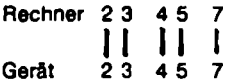
Reicht meistens für Terminals, Flußkontrolle ist dabei überflüssig.



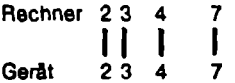
Manchmal für Terminals. Rechnerseitig wird Flußkontrolle durch die Brücke 4 – 5 simuliert.

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

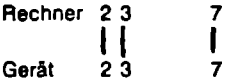
Vollständige Verbindung mit Flußkontrolle.



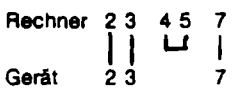
Einfacher Anschluß mit Flußkontrolle.



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. Ein Kanal heißt 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	zeichenorintiert, 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 CONST 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-Zeichencode") auf jedem Terminal die gleiche Semantik haben. Das heißt beispielsweise, daß der Code ""3"" 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.

Blockorientiere Ein – /Ausgabe

Blockorientiere 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 3 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 15) 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)').

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 110 angegeben.

cursor logic

PROC cursor logic (INT CONST offset, 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 sei, als

pre + code (offset + y) + mid + code (offset + x) + post

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

elbit cursor

PROC elbit cursor

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

baudrate

PROC baudrate (INT CONST channel, rate)

Zweck: Die serielle Schnittstelle des angegebenen Kanals wird auf die angegebene Baudrate eingestellt. Diese Prozedur muß nicht bei allen Rechnertypen vorhanden sein.

bits

PROC bits (INT CONST channel, number, parity)

Zweck: Die serielle Schnittstelle des angegebenen Kanals wird auf 'num-

ber' – Bits (7 oder 8) und die angegebene Parität eingestellt. Dabei bedeutet

0 : no parity
1 : odd parity
2 : even parity

Diese Prozedur muß nicht auf allen Rechnertypen vorhanden sein.

flow

PROC flow (INT CONST channel, type)

Zweck: Die Flußkontrolle des angegebenen Kanals wird auf 'type' eingestellt:

0 : keine Flußkontrolle
1 : RTS/CTS (Pin 4 und 5)
2 : XON/XOFF – Protokoll

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

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 (Code 100) zu. Diese Task kann daraufhin die nötigen Konfigurierkommandos ('link',...) ausführen.

Der standardmäßig ausgelieferte Konfigurations – Manager führt nur 'link' – Operationen durch. Falls weitere dynamische Operationen ('baud', 'bits', 'flow') notwendig werden, muß man ihn entsprechend modifizieren. Der Quellcode des entsprechenden Pakets befindet sich auf dem "conf" – Archiv.

Teil 3: ELAN – Programme

1. Wertebereiche und Speicherbedarf

INT – Objekte

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

REAL – Objekte

Jedes Datenobjekt vom Typ REAL belegt 8 Bytes im Speicher.

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

9.999999999999e + 126	größte REAL
0.000000000001	kleinste positive REAL mit x + 1.0 › 1.0
9.999999999999e – 126	kleinste positive REAL › 0.0
– 9.999999999999e – 126	größte negative REAL
– 9.999999999999e + 126	kleinste REAL

BOOL - Objekte

Jedes Datenobjekt vom Typ BOOL belegt 2 Bytes im Speicher.

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

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 4 INT	8
ROW 5 INT	16
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' beifügen muß.
- 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 dem Kommando 'halt'. Das 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 einer Operation bewirken, werden sie im folgenden zusammenfassend als Fehler bezeichnet.

Fur 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 'file does not exist' abgebrochen. Diesen Fehlerzustand fängt der Editor jedoch ab und versucht, eine neue Datei einzurichten. (Anmerkung: der Editor fragt 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(teile), 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 overflow' 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 letzte Fehlermeldung (eines 'error stop's) zugreifen. 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. 43).

```
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 overflow"
         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 über die Aufrufkette zurück kann auch über mehrere Stufen erfolgen, 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 Fehler auslösenden 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 'd' nicht ausgeführt wird. Da die Prozedur 'a' 'disable stop' gesetzt hat, werden die auf den Aufruf von 'b' folgende 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

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)

Merke: Fehler sind im EUMEL – System Aufrufe der Prozeduren 'errorstop', 'stop' oder das Betätigen der SV Taste und das 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' 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 adäquat 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.

error message

TEXT PROC error message

Zweck: Liefert die Fehlermeldung als Text. An Hand 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, 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'.

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.

Fehlercodes

Einige Fehlercodes sind bereits belegt:

0	kein Fehlercode spezifiziert
1	halt from terminal
2	stack overflow
3	heap overflow
4	INT overflow
5	DIV by 0
6	REAL overflow
7	TEXT overflow
8	too many DATASPACEs
9	subscript overflow

10	subscript underflow
11	alias error at dataspace access
12	undefined INT
13	undefined REAL
14	undefined TEXT
15	storage overflow
16	code block unreadable
17	wrong opcode
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

THESAURUS
TYPE THESAURUS
Zweck: Bezeichnet Thesaurus – Datenobjekte

:=

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

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: Fall 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
Zwech: 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:

* invalid name

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:

* invalid name
* thesaurus overflow

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 Indices 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:

* invalid name

PROC rename (THESAURUS VAR t, INT CONST index, TEXT CONST new)

Zweck: Ändert im Thesaurus 't' den Namen des durch 'index' identifizierten Eintrag in 'new'.

Fehlerfall:

* invalid name

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 (auch 'myself') 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'.

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) : SOME ALL task

THESAURUS OP SOME (TEXT CONST file name) : SOME ALL file name

+

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, right)

Zweck: Liefert die Differenzmenge. Achtung: Die Differenzmenge enthält keine Namen mehrfach.

/

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.

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'.

save

PROC save (THESAURUS CONST thesaurus)
Zweck: Schickt alle aufgeführtren Dateien zur Vater – Task.

PROC save (THESAURUS CONST thesaurus, TASK CONST manager)
Zweck: s.o.

forget

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

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 'manager' – Task.

insert

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

fetch all

PROC fetch all (TASK CONST manager)
Zweck: Holt alle Dateien vom 'manager'.

save all

PROC save all (TASK CONST manager)
Zweck: Schickt alle eigenen Dateien zum 'manager'.

Beispiele:

```
save (ALL myself)
forget (ALL myself)
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. Zusätzlich ist er temporär ausgeschaltet, wenn kein Terminal angekoppelt ist.

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 wir 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.

command handler

PROC analyze command (TEXT CONST command list, command line,
INT CONST additional 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 'additional 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 werden als Niltext geliefert. Ist das Kommando vorhanden, aber die Anzahl der Parameter stimmt nicht, wird der negative Hauptindex geliefert; ist es vollkommen unbekannt oder ist die Kommandoeingabe 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 additional 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 ("unknown command")
                  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. Z.B. muß das 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 automtisch auf FALSE gesetzt. So wird erreicht, daß diese Prozedur in den neu eingerichteten Sohn und Enkeltasks genau beim ersten Aufruf FALSE liefert.

Beispiel:

```
PACKET stack DEFINES push, pop:

INITFLAG VAR this packet := 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 (this packet)
    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 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'.

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.

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.

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 internewn 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:

* task not existing

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!

exists

BOOL PROC exists (TASK CONST task)

Zweck: Informiert, ob die angegebene 'task' noch existiert. Der eigene Taskkatalog wird dabei aktualisiert.

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

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

Fall die Task "hugo" nicht existiert, führt nämlich schon der Aufruf 'task ("hugo")' zum 'errorstop ("task not existing").

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 100, 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 PUBLIC-Task.

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:

* task not existing

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 mit dem Einverständnis beider – des Senders und des Empfängers – möglich. 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 anderern 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 verwandt 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	"destination task not existing"
– 2	"destination task not ready to receive"

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 Manger 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 zwei 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.

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 :

* task directory overflow

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 :

* task directory overflow
* incorrect task name (* "" oder LENGTH > 100 *)
* duplicate task name

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:

* invalid channel
* channel not free

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. 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:

* not parent

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 aufwendiger 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 Taskname '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:

* duplicate task name
* incorrect task name

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 autonom (üblich für Manager-Tasks). Wenn jetzt ein 'continue'-Supervisor-Kommando auf diese Task von einem Terminal aus gegeben wird, wird die 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 angegebenen 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. Andere Tasks sollten sie normalerweise nicht anwenden, um dem Scheduling nicht entgegen zu wirken.

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.

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

end

PROC end (TASK CONST task)

Zweck: Die angegebene Task wird 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.

Achtung: Diese Operation erfordert 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: Näheres zur Bedeutung der 'prio' findet man im Kapitel 'Scheduling'.

set date

PROC set date

Zweck: Datum und Uhrzeit können im Dialog gesetzt werden (Form wie bei dem Start des Systems) .

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 ist ('command dialogue (TRUE)'). 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 entgegen zu wirken.

3. 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 Dateimanager, 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 'erase task' 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 4000 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 *Standardarchiv*.

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.

Hinweis: Man darf die zwar große, aber doch endliche Dateikapazität nicht vergessen. Nachdem das Logbuch mit 4073 Sätzen voll ist, werden weitere Protokollsätze ignoriert. Die Datei "logbuch" sollte deshalb von Zeit zu Zeit gelöscht werden ('erase' bzw. 'forget'). Wird das Logbuch nur vom EUMELmeter benutzt, können bis zu ca. 670 Betriebsstunden protokolliert werden.

Teil 6: Drucker und Textverarbeitung

1. Standard – Drucker vom Archiv generieren

Auf dem Archiv 'STD' befinden sich neben dem 'std printer' auch die Generatordateien

std printer generator/S

zum Generieren eines Standard – Druckers im Single – User, sowie

std printer generator/M

zum Generieren eines Standarddruckers im Multi – User.

Die Generatoren kennen nur einen ganz einfachen Drucker:
- jeweils ein Typensatz (äquidistant)
- einzige beim Drucker vorausgesetzte Steuerzeichen sind ‹ CR › ‹ LF ›.

Im Single – User wird der Drucker durch folgende Kommandos generiert:

```
archive ("std"); fetch ("std printer generator/S", archive);
run
```

Im Multi User muß der Drucker in einer Task mit dem Namen 'PRINTER' generiert werden, die ein Sohn des SUPERVISOR's sein sollte.

Dies geschieht durch folgendes Supervisor – Kommando:

```
begin ("PRINTER", "SUPERVISOR")
```

und dann die folgenden Monitor – Kommandos:

```
archive ("std");
fetch ("std printer generator/M", archive);
run
```

Am Schluß jeder Generierung wird der Druckerkanal erfragt.

Das Umsetzen von Zeichen im EUMEL – Code in den entsprechenden Druckercode (Umlaute, ß usw.) geschieht mit Hilfe des Konfigurators, indem der Drukkerkanal entsprechend belegt wird (siehe "Konfigurator").

2. Druckeranpassung

Aufbau des Drucksystems

Die Generierung eines Drucksystems mit eigenem Hardware Interface erfolgt durch Insertieren in folgender Reihenfolge:

Single User	*Multi User*	
hardware interface	hardware interface	
eumel printer	*eumel printer*	
elan lister	elan lister	(optional)
printer/S	spool manager	
	printer/M	

Dabei haben die einzelnen Pakete folgende Bedeutung:

– hardware interface
  Hier werden die elementaren Routinen zur Ausgabe auf dem Drucker bereitgestellt. Dies Paket dient als Schnittstelle zwischen der Druckerhardware und dem 'eumel printer' und ist deshalb im allgemeinen auch hardwarespezifisch programmiert.

- eumel printer
  Der 'eumel printer' verarbeitet die mit Hilfe der Textkosmetik erstellten Dateien. Dabei ruft er die entsprechenden Prozeduren des 'hardware interface' auf.

- elan lister
  Die Installation des 'elan lister's ist optional. Bei insertiertem 'elan lister' werden Texte, die mit einem Kleinbuchstaben beginnen, als ELAN-Programme angesehen (Refinementname) und entsprechend aufbereitet ausgegeben.

- spool manager
  Der 'spool manager' ist nur im Multi-User zu verwenden. Er verwaltet in der Task PRINTER die eintreffenden Druckaufträge und leitet diese an den Drukker weiter.

- printer/S
  Dieses Paket ist nur im Single-User zu verwenden. Es stellt die zum Drucken benötigte 'PROC print (TEXT CONST name)' zur Verfügung. Diese schaltet auf den Druckerkanal um und übergibt die zu druckende Datei an den 'eumel printer' bzw. 'elan lister'.

- printer/M
  Dieses Packet startet den Druckerspool, erfragt den Druckerkanal und richtet an diesem den Drucker-Worker ein.

Von den oben beschriebenen Paketen ist das Hardware-Interface speziell für die einzelnen Drucker zu erstellen, während die übrigen Pakete nicht geändert zu werden brauchen und von der EUMEL-Gruppe gewartet werden.

Definition des Drucker-Interfaces

Um den 'eumel printer' an eine spezielle Druckerhardware anzupassen, müssen bestimmte Prozeduren zur Verfügung gestellt werden. Der auf dem Archiv STD ausgelieferte 'std printer' stellt ein solches Hardware Interface dar, das auf fast jeder Druckerhardware lauffähig ist. Allerdings nutzt er keine der oft verfügbaren besonderen Druckerfähigkeiten (mehrere Typen, besseren Randausgleich, Pro-

portionaldruck usw.) aus. Will man alle Eigenschaften des eigenen Druckers nutzen, muß man ein speziell darauf zugeschnittenes Hardware Interface programmieren. Es muß dem Eumel Drucker folgende Prozeduren zur Verfügung stellen:

change type

PROC change type (TEXT CONST name of type)

Zweck: Der Schrifttyp wird auf den angegebenen Typ umgeschaltet. Diese Umschaltung kann auch mitten in einer Zeile auftreten. Die Typnamen sind installationsspezifisch (d.h. frei wählbar). Der Type "" (Niltext) ist allerdings immer der Standardtyp, sofern keine Fonttabelle geladen wurde (beim Standarddrucker ist dies nicht erforderlich). Alle unbekannten Schrifttypen sollten auf den Standardtyp abgebildet werden.
Alle durch das Kommando 'on' gesetzten Einstellungen werden zurückgesetzt.

lf height of current font

REAL PROC lf height of current font

Zweck: Abstand zwischen 2 Zeilen bei einfachem Zeilenvorschub (im Normalfall 2.54 / 6.0). Dieser Wert kann mit Hilfe der Fonttabellen berechnet werden.

limit

PROC limit (REAL CONST x)

Zweck: Die Zeilenlänge wird dem Hardware Drucker als reelle Zahl in cm angegeben. Dieser Wert dient der 'block' – Routine zur Berechnung der Wortzwischenräume.

line

PROC line (REAL CONST proposed lf)

Zweck: Es sollen 'proposed lf' Zeilenvorschübe durchgeführt werden. Als Anforderung ('proposed lf') können beliebige REAL – Werte auftreten. Die Differenz zwischen dem tatsächlich durchgeführten und dem verlangten Vorschub muß aufbewahrt werden, damit entsprechende Korrekturen (z.B. bei 'new page') – falls notwendig – durchgeführt werden können. Falls ein 'proposed lf' hardwaremäßig nicht möglich ist, sollte auf den nächstmöglichen Wert abgerundet werden. 0.0 sollte nur dann gewählt werden, wenn 'proposed lf' tatsächlich diesen

Wert hat. So sollte bei einem Drucker, der nur ganze Zeilenvorschübe ausführen kann, 1.5 auf 1.0 abgebildet werden, 0.5 aber nicht auf 0.0 sondern auf 1.0.

material

PROC material (TEXT CONST name of material):

Zweck: Es soll auf das angegebene Papiermaterial umgeschaltet werden. Das Kommando kann nur zu Beginn eines Druckvorgangs auftreten. Die Materialnamen sind installationsspezifisch, d.h. frei wählbar. Nur "" (Niltext) gibt immer das Standardmaterial an. Alle unbekannten Materialien sollten auf das Standardmaterial abgebildet werden.

new page

PROC new page

Zweck: Es soll auf den Beginn der nächsten Seite positioniert werden. Der Druckanfang soll auf der durch das 'start'-Kommando festgelegten Position liegen.
Falls die aktuelle Position gerade durch diese Werte beschrieben wird, soll kein Seitenvorschub (keine Leerseite) erzeugt werden. Befindet sich die aktuelle Position noch vor der durch das 'start'-Kommando festgelegten Position, so wird auf der aktuellen Seite bis zur Startposition vorgeschoben.

off

PROC off (TEXT CONST attribute)

Zweck: Schaltet die mit 'on' gesetzten Modifikationen aus.

on

PROC on (TEXT CONST attribute)

Zweck: Schaltet die Modifikation 'attribute' des aktuellen Schrifttyps ein. Zur Zeit ist unterstrichen (underline), fett (bold), kursiv (italic) und der Druck von weiß auf schwarz (revers) möglich. Bei der Analyse der Konstanten 'attribute' wird nur der erste Buchstabe untersucht. Ist eine Modifikation auf der betreffenden Druckerhardware nicht möglich, so wird sie ignoriert.

papersize

PROC papersize (REAL CONST weidth, length)

Zweck: Liefert die Werte für Länge und Breite des verwendeten Papiers an den Hardware – Drucker. Hierdurch ergeben sich Möglichkeiten zur Überprüfung der durch 'limit' und 'pagelength' gesetzten Einstellungen.

print text

PROC print text (TEXT CONST content, INT CONST printmode)

Zweck: Der übergebene Text soll gedruckt werden. 'content' kann dabei eine ganze Zeile, aber auch nur einen Teil einer Zeile umfassen. 'printmode' informiert über linksbündigen Druck und Blocksatz, sowie rechtsbündigen Druck und Zentrieren unter Verwendung von Tabulatoren.

printmode	Wirkung
0	linksbündig
1	rechtsbündig
2	zentrieren
3	Blocksatz

Wurde mit 'x pos' eine Positionierung in X – Richtung vorgenommen, so wird der Text 'content' dieses und aller folgenden Aufrufe von 'print text' bis zum nächsten Aufruf von 'xpos', 'ypos' oder 'line' gesammelt und erst dann gemäß printmode bzgl. der alten Positionierung in X – Richtung ausgegeben. Wurde eine Positionierung in X – Richtung vorgenommen und ist Blocksatz gefordert, so wird 'content' auf die Länge (limit – xpos) geblockt und dann linksbündig bzgl. der Positionierung ausgegeben.

Wurde keine Positionierung in X – Richtung vorgenommen, so kann 'printmode' nur die Werte 0 oder 3 annehmen. 'content' wird dann ab der aktuellen Position linksbündig bzw. auf 'limit' geblockt ausgedruckt.

printer cmd

PROC printer cmd (TEXT CONST cmd)

Zweck: Gibt die Anweisung 'cmd' als direkte Druckersteuerungsanweisung an den Drucker weiter. Solche Anweisungen werden benötigt, um spezielle Druckereigenschaften auszunutzen, die im EUMEL-Drucker nicht realisiert sind.

reset printer

PROC reset printer

Zweck: Der Drucker soll in den Grundzustand überführt werden (Standardschrifttyp, Standardpapiergröße, usw.). Die Prozedur wird zu Beginn jedes Druckvorgangs aufgerufen.

start

PROC start (REAL CONST x, y)

Zweck: Durch die cm-Angaben x und y wird die obere linke Ecke des Schreibfeldes festgelegt (linker Rand: x cm, oberer Rand: y cm). 'new page' positioniert dann immer auf die durch diese Prozedur festgesetzte obere linke Ecke.

x factor per inch

INT PROC x factor per inch

Zweck: Gibt die Anzahl der minimalen Schritte pro inch des aktuellen Schrifttyps in x-Richtung an. Dieser Wert kann mit Hilfe der Fonttabellen berechnet werden.

x pos

PROC x pos (REAL CONST wert)

Zweck: Positioniert in der aktuellen Zeile auf 'wert' cm vom linken Rand des Schreibfeldes.

y pos

PROC y pos (REAL CONST wert)

Zweck: Positioniert vertikal auf 'wert' cm vom oberen Rand des Schreibfeldes aus.

3. Erstellung der Schrifttyp – Datei

Der EUMEL – Drucker sowie 'lineform/autoform' und 'pageform' ("Textkosmetik") sind standardmäßig auf eine Schrift mit 10 Zeichen/Inch und 6 Zeilen/Inch eingestellt. Der Name dieser Schrift ist "". Die Schrifttyp – Datei dient dem EUMEL – Drucker und der Textkosmetik, um einen oder mehrere andere Schrifttypen einzustellen.

Sie müssen den voreingestellten Zeichensatz nur ändern, wenn

- der Drucker Ihrer Installation über einen anderen als den voreingestellten Zeichensatz (z.B. 12 Zeichen/Inch) verfügt;

- man mit lineform mehrere Zeichensätze bearbeiten will, weil der zur Verfügung stehende Drucker mehrere Zeichensätze abbilden kann oder weil eine Datei formatiert werden soll, die auf einer anderen Installation mit anderen Zeichensätzen
  gedruckt werden soll.

Dies erfolgt mit dem Aufruf der Prozedur

```
PROC load font table (TEXT CONST schriftdatei)
```

und kann – soweit erforderlich – mehrmals wiederholt werden. Sie müssen 'load font table'

a) in der Task PRINTER geben, um den EUMEL – Drucker mit den Schriften zu laden. Beachten Sie, daß sie für eine Proportional – Schrift den EUMEL – Drukker anpassen müssen (der Standard – Drucker kann keine Proportionalschriften) oder einen vorgefertigten anderen Drucker übernehmen müssen.

b) in der Task PUBLIC, um 'lineform/autoform' und 'pageform' mit den Schriften zu laden. Die neuen Schriften gelten dann für jede neu eingerichtete Task. Bereits vorher vorhandene Tasks haben noch die voreingestellte Schrift bzw. vorher geladene Schriften.

c) Sie können die Schrifttyptabelle auch in einer anderen Task laden. Dann gelten die neu eingestellten Schriften nur in dieser Task (bzw. deren Sohn – Tasks). Das ist manchmal notwendig, wenn Dateien formatiert werden sollen, die auf anderen Druckern gedruckt werden sollen.

Die eingestellten Schriften in 'lineform' werden bei jedem Aufruf von 'load font table' überschrieben.

'lineform'/'pageform' und der 'eumel drucker' müssen die Breite, Höhe u.a.m. von Zeichen wissen. Diese werden in der Schrifttyptabelle angegeben. Aus Effizienzgründen wird intern aber nicht mit REALs, sondern mit INTs gerechnet. Zur Berechnung der (INT –) Breiten wird ein Umrechnungsfaktor benutzt. Dieser ist standardmäßig auf 250 gesetzt, um auch besonders breite Schriften (mehr als 1 cm) verarbeiten zu können. Allerdings treten dann Ungenauigkeiten in Form von Rundungsfehlern auf. Dies kann vermindert werden, indem man den Faktor erhöht mit

```
factor (500)
```

In der Datei 'schriftdatei' müssen die einzustellende(n) Schrift(en) für Drucker und/oder 'lineform' zeilenweise nach folgender Konvention stehen:

‹ Schriftname › # ‹ fest o. prop › ‹ Blankbreite › ‹ Lf › ‹ factors › [Schritt] [nr]
‹ Zeichen › ‹ Zeichenbreite › [Ersatzzeichen]]

Die Angaben in spitzen Klammern müssen vorhanden sein, während die in eckigen Klammern optional sind. Jede Angabe muß durch (mindestens) ein Leerzeichen von einer folgenden Angabe getrennt werden. Die Angabe der zweiten Zeile kann mehrmals erscheinen. Mehrere Schriftsätze können hintereinander angegeben werden.

Der erste Schriftsatz der Datei ist der automatisch voreingestellte Schrifttyp von lineform, pageform und EUMEL – Drucker. Das ist der Standard – Schriftsatz, den man erhält, wenn

– keine 'type' – Anweisung in einer von 'lineform' zu bearbeitenden Datei oder einer vom Drucker zu druckenden Datei angegeben wird.

– in einer 'type' – Anweisung ein unbekannter Schriftname angegeben wurde.

Dabei bedeuten:

– Schriftsatzname der Name des Schriftsatzes (ohne Leerzeichen!) (also der, den man 'type' Kommando nachher angeben muß). Beispiele:
#pica# (* name 'pica' *)
#elitedeutsch# (* Name kann im 'type' – Kommando 'elite deutsch' geschrieben werden *)
#trium60# (* *Faktoren und der Schriftname hängen zusammen* *)
Existiert eine Schrift in unterschiedlichen Größen, muß mehr als ein Faktor angegeben werden. In diesem Fall muß der erste der Faktoren am Ende des Schriftnamens in Form von Ziffern angegeben werden. Ein Schriftname ohne Ziffern am Ende bedeutet also, daß es von dieser Schrift keine unterschiedlichen Größen gibt.

– *fest oder prop*
Angabe, ob es sich um einen äquidistanten Schriftsatz (Angabe: fest) oder einen Schriftsatz handelt mit einer Proportional – Schrift (prop).

– Blankbreite
die Breite in cm des Leerzeichens des betreffenden Schriftsatzes
(z.B. 0.254 für einen Zeichensatz mit 10 Zeichen/Inch
0.212 " " " " 12 " ")

– Lf
Zeilenvorschub in cm für den angegebenen Schriftsatz. Der Drucker (und page–form) bewegt das Papier inhorizontaler Richtung um diesen Betrag, wenn eine Zeile mit dem ‹ Schriftsatzname › geschrieben wird. Lf sollte so gewählt werden: Höhe des größten Zeichens + 10%.

– factors
Ein Schriftsatz kann in mehreren Größen vorhanden sein. Die geklammerten und durch Komma und blank getrennten factors – Angaben geben diese Unterschiede in etwa an. Die Faktoren sind in der Regel 'points' – Angaben. Beispiel:
(10, 15)
Die Faktoren hängen auf folgende Weise mit den Schriftnamen zusammen:
Existiert eine Schrift mit unterschiedlichen Größen, muß eine Grössenangabe an den Namen angehängt *werden (in Form von Ziffern). Die Ziffernangabe im* Namen muß mit der jeweilig ersten Faktor – Angabe übereinstimmen. Beispiel:
#trium60# ... (60, 72, 80) ...
#trium72# (72, 60, 80) ...
#trium80# ... (80, 60, 72) ...

Die Faktoren dienen zur automatischer Schrifttypberechnung (z.B. in Formeln, um eine kleinere oder grössere Schrift gleichen Typs zu finden). Wird nur ein Faktor angegeben, existieren keine anderen Grössen der gleichen Schrift.

- Schritt
  Anzahl Schritte pro Blank der kleinsten horizontalen Bewegung, die ein Drucker leistet.

- nr
  Gibt die Nummer des Schrifttyps für einen bestimmten Drucker an.

Die folgenden Angaben sind erforderlich, wenn es sich um einen Proportionaltyp handelt und/oder wenn eine Zeichenumsetzung vom Drucker vorgenommen werden muß:

- Zeichen und Zeichenbreite
  Sind bei Proportionalschriften diejenigen Zeichen, deren Breiten in cm von der Breite des eingestellten Blanks abweicht. Beispiel:
  A 0.254
  B 0.212
  C 0.230
  :
  Z 0.230

- Ersatzzeichen
  gibt das Zeichen an, welches für ‹ Zeichen › gedruckt werden soll. Beispiel:
  [0.254) ("Für das Zeichen '[' wird das Zeichen ')' gedruckt ")

Beispiele:

a) Einstellen eines Schriftsatzes mit Namen "" und 10 Zeichen/ Inch (Äquidistant):

```
## fest 0.254 0.42333 ( 10 )
```

b) Einstellen eines Schriftsatzes mit Namen "" und 12 Zeichen/ Inch (Äquidistant):

```
## fest 0.212 0.42333 ( 12 )
```

c) Einstellen von zwei äquidistanten Schriftsätzen:

```
## fest 0.254 0.42333 ( 10 )
#elite# fest 0.212 0.4233 ( 12 )
```

d) Einstellen eines Proportionaltyps:

```
#propschrift# prop 0.254 0.4233 ( 10 )
A 0.260
B 0.270
C 0.270
...
Y 0.230
```

e) Einstellen von zwei äquidistanten und zwei proportionalen Schriftsätzen (mit 'pica' als erstem, also Standardtyp):

```
#pica# fest 0.254 0.4233 ( 10 )
#elite# fest 0.212 0.4233 ( 12 )
[ 0.212 (
] 0.212 )
#prop# prop 0.212 0.4233 ( 10 )
A 0.217
B 0.23
...
Z 0.22
#nocheinprop# prop 0.212 0.4233 ( 10 )
A 0.25
B 0.25
...
Ä 0.237
```

f) Einstellen eines proportionalen Schriftsatzes in zwei unterschiedlichen Schriftgrössen:

```
#test10# prop 0.254 0.4233 ( 10, 12 )
A 0.25
N 0.25
...
#test12# prop 0.254 0.4233 (12, 10)
```

Teil 7: 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 (siehe Graphik – Editor).

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 CONST x, y, z)

Aktion: Der aktuelle Stift muß gehoben zu den Rasterkoordinaten 'h,v', die den 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 aktuell 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. Spooler

Das Paket "spool manager" befindet sich auf dem mitgelieferten Archiv. Es stellt folgende Prozedur zur Verfügung:

spool manager

PROC spool manager (PROC worker)

Zweck: Die Task, in der die Prozedur aufgerufen wird, wird zur Spooler – Task (Näheres siehe Benutzerhandbuch). Es wird automatisch eine Worker – Task als unbenannter Sohn ("–") eingerichtet und mit der übergebenen 'PROC worker' gestartet. Dieser Worker kann vom Spooler aus durch Spooler Kommandos (s. Benutzerhandbuch) gelöscht und wieder neu eingerichtet werden.
Der Worker erbt vom Spooler beim Start die Paketvariablen mit ihren aktuellen Werten. So kann er sich z.B. über einen zugeordneten Kanal (für einen Drucker oder Plotter als Worker) informieren.

Die Worker – Task kann sich neue Datenräume aus der Spooler Warteschlange folgendermaßen holen:

```
LET fetch code = 11 ;
call (father, fetch code, ds, reply)
```

Bei 'reply' = 'ack' (0) wurde der nächste Datenraum geliefert. Eine andere Rückmeldung (*error nak*) tritt *nur dann auf, wenn diese Anforderung nicht vom* Worker des Spoolers, sondern von einer anderen Task erfolgt.

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 – den 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 physikalischer 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 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.

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 wird die vom letzten 'in' Aufruf bzw. seit der Assoziierung bis jetzt auf dem Kanal eingetroffenen Daten geliefert. Bei 'niltext' liegen keine Eingabedaten vor.

open

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 not existing
- channel not free

out

PROC out (FCHANNEL VAR f, TEXT CONST message)

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

PROC out (FCHANNEL VAR f, DATASPACE CONST file space)

Zweck: Der übergebene Datenraum muß eine Textdatei sein (Struktur eines FILEs haben). Er wird komplett auf dem Kanal 'f' ausgegeben.

PROC out (FCHANNEL VAR f, DATASPACE CONST file space,
TEXT CONST handshake char)

Zweck: s.o. Bei der Ausgabe wird nach jeder Zeile auf eine Quittung gewartet. Es werden alle Zeichen weggelesen, bis der spezifizierte 'handshake char' ankommt. ""0"" ist nicht als 'handshake char' zugelassen.

Beispiele:

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

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

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

Stichwortverzeichnis

Autoren:

Jochen Liedtke, Dietmar Heinrichs, Rainer Hahn

Kontaktadresse:

HRZ der Universität Bielefeld
Postfach 8640

4800 Bielefeld 1

Umschlaggestaltung:

Hannelotte Wecken

Druck:

Zentrale Vervielfältigungsstelle der Universität Bielefeld
im Februar 1984

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.