User Manual

Part 1: Word Processing

June 1, 1987

© GMD 1987

Alle Rechte vorbehalten. Insbesondere ist die Überführung in maschinenlesbare Form sowie das Speichern in Informationssystemen, auch auszugsweise, nur mit schriftlicher Einwilligung der GMD gestattet.

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the GMD.

Authors: Monika Fey – McClean
Konrad Klöckner
Werner Metterhausen et al.

Translated by: Ursula Bernhard
Monika Fey – McClean

This text was edited with the EUMEL word processing system and printed with the AGFA printing system P400.

Gesellschaft für Mathematik und Datenverarbeitung mbH

Bereich/Location Birlinghoven
Schloß Birlinghoven
Postfach 1240
D – 5205 Sankt Augustin 1

Telefon (02241) 14 – 1
Telex 8 89 469 gmd d
Telefax (02241) 14 28 89
Teletex 2627 – 224135 = GMDVV
Bildschirmtext *43900#

PART 1: Introduction

PART 2: The Supervisor

PART 3: The Monitor

PART 4: The Editor

PART 5: Text cosmetics and printing

PART 6: Special features

Appendix

Index

PART 1: Introduction

1.1. Preface

This book gives an introduction to the use of the operating system EUMEL as word processing system. The book is thus intended for all those who want to use the easy – to – learn EUMEL system for the production of texts of any type and volume.

This introduction requires no previous knowledge, neither of computers in general, nor of EUMEL in particular. Newcomers in the field of data processing should read this first part, which deals briefly with "mere theory", at least twice:

- The first reading should serve a rough orientation only. For a better understanding of the following chapters, you should have already heard about some terms that will partly make sense only to the more experienced user.

- The second reading will be recommendable when you have found your feet. After a few days, when the handling of the EUMEL system has already become somewhat more familiar to you, you should reread the explanations against the background of the gained experience. Only then, some terms will become really understandable and some uncertainty will be removed.

The following parts of the book will provide further instructions reaching from the first steps towards the EUMEL system up to a detailed description of word processing. You should carefully go through all examples contained in parts 3 and 4 and try them at the video terminal. Select from part 5 those examples you consider to be particularly important. You will see that an increasing routine in handling EUMEL enables you to add further features from the EUMEL word processing facilities.

What is an operating system?

An operating system is a collection of programs which enable a computer user to work with the computer. The programs of the operating system interconnect the components of a computer, the hardware, and the application programs of a user.

All programs which fill this enormous gap and secure, e.g., that the command:

print ("this letter")

has in fact the effect that the letter just written at the video terminal is transmitted to the printer to be printed there, are so-called system programs, i.e. parts of the operating system.

This user manual for the operating system EUMEL will stepwise explain the commands you are able to use for the utilization of the EUMEL word processing facilities and therefore present a part of the operating system to you.

Is EUMEL different from the rest?

Yes. The operating system EUMEL (Extendable multi User Microprocessor ELan system) is essentially different to other systems as its expanded name implies: "extendable multi-user microprocessor ELAN system".

While other microprocessor- or personal-computer-oriented systems support only one user in his/her work, EUMEL allows several users a concurrent computer utilization. Of course, EUMEL works just as well for a single user. EUMEL provides, however, the possibility of connecting further video terminals to the computer, thus allowing several users a concurrent utilization **without** any cost for additional software.

Secondly, EUMEL is hardware-independent, that is, no matter who is the manufacturer of your computer, operation and command language will always be the same. Even diskettes written by an XY computer can be read by an ABC computer which is by no means a matter of course.

A further special feature of the EUMEL system makes everybody working with it happy: EUMEL is consistently written in the programming language ELAN. Even if you are not (yet) able to program, ELAN makes life easier for you by enabling you to write what you really mean: a file containing a business letter complete and ready for printing is, for example, not called:

$txt.prt

but:

Offer for Messrs. Miller 1.7.86

A further important difference will become clear to you if you know another operating system: the EUMEL word processing is not an additional program with its own specific command language which, if needed, must be loaded, but is available at keystroke in the true sense of the word.

1.2. Some important terms

- **TASK** A task is an independent process within a EUMEL system. It belongs either to the management of the EUMEL system or to a user. By allocating a specific work area to every user, an uncontrolled access to foreign data is avoided. A task has a name that is used to address the task. An EUMEL system consists of several tasks.

  The EUMEL task system can be compared to an office building: it consists of many rooms and a room (= task) is either a normal workroom or the office of an executive or a workshop where services for others are provided.

  Such a ranking is illustrated by the following task system, the notes written *in italics* show the names of the "departments" of a comparable office:

```
SUPERVISOR (* room administrator *)
    -
    -
    SYSUR (* foreman *)

        ARCHIVE (* archivist *)

        configurator (* electrician *)

        OPERATOR (* janitor *)

            shutup (* night-watchman *)

UR (* supervisory board *)

    PUBLIC (* departmental head *)

        Brown  (* employee *)
        Miller (*    "     *)
        Smith  (*    "     *)
```

A task thus represents a workroom for a EUMEL user if we use the above comparison. As a EUMEL user you create your task yourself by entering the command 'begin ("task name")'.

After you have given this command once, this task is available under the name chosen by you. You work within the task (i.e. analogously in the workroom) –, in particular, you set up files (= folders). Files only exist within a task.

Tasks are managed by the SUPERVISOR, it regulates the access to tasks. In order to leave your task, send the command 'break' to the SUPERVISOR, to reenter the task, enter the command 'continue ("task name")'.

- **FILE** A file is a set of data belonging together. A file in a task corresponds to a folder in a workroom. A task may contain up to 200 files. Each file in a task has its own name, files of the same name may exist in different tasks. A file is subdivided into lines.

  For processing a task at the video terminal, the file must be accessed via the Editor: 'edit ("file name")' enables you to revise the file contents at the terminal (see Parts 4 and 5).

- **COMMAND** A command is an order to the computer saying that a work is to be done. Which commands you can give to the computer at a specific point of time depends on the current "command level" you are working at. In general,

  - commands at Supervisor level affect the task system;

  - commands at Monitor level affect your own task or files;

  - commands at Editor level affect lines, words or individual characters of the current file. After a short time of familiarization, you will easily identify the level you are working at (see Part 3).

For some commands you must specify not only what is to be done, but also the object to which it is to be done. Such an addition to a command is called a parameter .

			command	parameter
			|	|
Example:	create new task	=	begin	("task name")
	print file	=	print	("file name")
	Find the word END	=	down	("END")

Parameters are put in parentheses and separated by commas where appropriate. Text parameters are additionally enclosed in double quotes.

A command may require no or many parameters; the description of the commands contained in this Manual will show all possibilities.

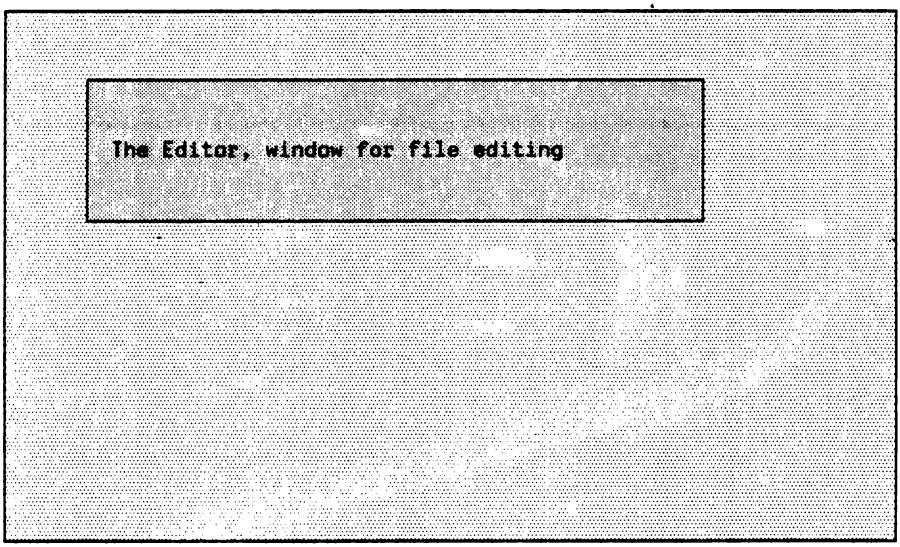
- **SUPERVISOR** Special task for supervising a EUMEL system. By means of the Supervisor command you can demand services from this task, i.e. create a new task, continue a task and obtain some information.

- **MONITOR** Recipient of commands in a task. Any work in the EUMEL system is done within a task. To a large extent working with a computer consists of calling programs by means of commands. The recipient of these commands in a task is the Monitor. The Monitor is identifiable by a line showing 'enter command'. In this line commands and parameters can be entered, if required.

- **ARCHIVE** Special task for managing the disk drive. Since, for longer-term data storage and for additional data security, files are written onto diskettes, the EUMEL system provides a special task which facilitates handling and secures an exclusive access to the drive.

- **EDITOR** Program for file editing at the video terminal. The program is started by means of the (Monitor) command 'edit' followed by the name of the requested file as parameter.

  Since a screen layout is normally restricted to 24 lines of 80 characters' line width, the Editor can be regarded as a window which is moved over a considerably larger file and which enables you to edit the displayed part.



1.3. Notation used in this book

The following text stepwise introduces you in the handling of the system. In this manual all commands and activities are illustrated by examples which you should try directly at your computer terminal.

Please observe the following notation rules:

- Some keys of a computer keyboard have a special meaning. These so-called function keys and special key combinations are explicitly represented as keys:

  SV

  ESC e

  CR

- Everything you are supposed to write or read on the screen of your terminal is enclosed in a box representing a screen.

  Example:

```
enter command :
edit ("my file")
```

- The notation used in the Manual follows the conventions of the programming language ELAN, i.e. the language in which all operating system programs are written. The following specifics are to be noted:

  1. Commands are always written in lower case characters.

  2. File names and the like are put in parentheses and double quotes. At places where a file name is to be inserted, this Manual shows the word *'file name'*; replace this 'file name' by the actual name you are free to choose.

  3. Any special term or example contained in a normal text is enclosed in simple quotes.

  To sum it up: The command 'edit' requires a file name as parameter. Please choose a name and enter 'edit ("file name")'. If you have chosen the name *"business letter"* type:

```
edit ("business letter")
```

and by means of the CR send the command to the Monitor for execution:

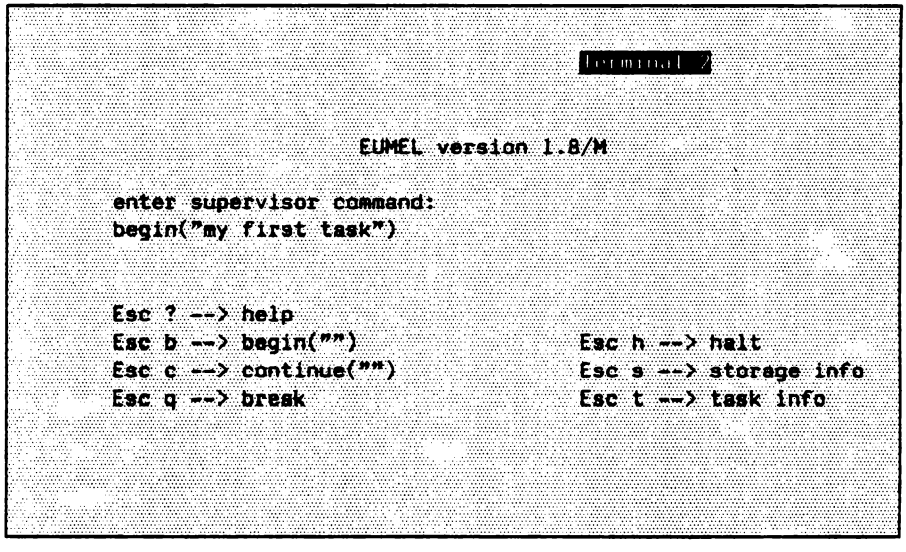
```
enter command :
edit ("business letter")
```

In the following the entry of CR as "mechanism" effecting the execution of commands will no longer be mentioned in particular.

1.4. Conditions

Apart from the computer, the complete installation of a EUMEL system on this computer is required for all activities described in the following.

The description of a system installation is contained in the Appendix. Furthermore, upon entering SV or CNTL b (simultaneously), your computer should display the so-called basic menu of EUMEL and accept Supervisor commands.



For further information on the installation of a EUMEL system, please refer to the Appendix.

The function keys of the EUMEL system

Please refer to the installation instructions of your specific equipment for locating the EUMEL function keys on the keyboard.

← → ↑ ↓	positioning keys
SHIFT	shift key
CR	input/end – of – line key
HOP	booster key
RUBOUT	delete key
RUBIN	insert key
TAB	tab key
MARK	mark key
ESC	command key
SV	supervisor key
STOP	stop key
PROCEED	proceed key

Task organization

For a better understanding of system handling, you should try to get an idea of how the system parts are organized.

The individual tasks of a EUMEL system "do not float unsupported in the air", but they are organized in a tree structure:

```
SUPERVISOR
    -
    SYSUR
        configurator
        OPERATOR

        ARCHIVE
UR
        PUBLIC
            Brown
            Miller
            Smith
```

The system consists of two branches which lie next to one another, i.e. the system branch with the root SUPERVISOR and the user branch with the root UR.

The system branch provides you with privileged services, the user branch represents the normal working environment.

All tasks of the EUMEL system being below these roots have at least one predecessor, i.e. there is a "father – son relation" between all tasks of the system.

In general, files can be sent to the father task and fetched from it without entering any special command. This is, however, not possible for any other task.

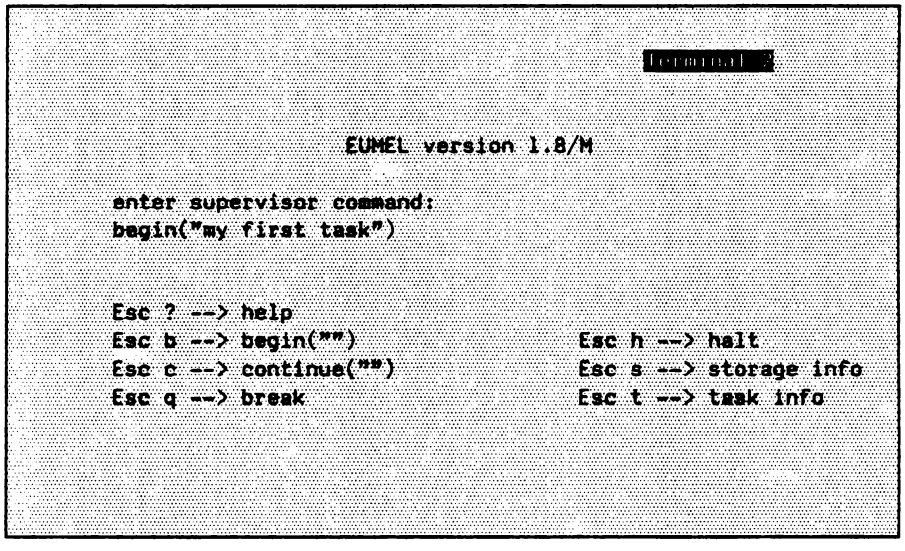
'Miller' may send a file to 'PUBLIC' and 'Smith' may fetch it from there, but a direct transmission from 'Miller' to 'Smith' is usually impossible.

As to terminology: every task via which this kind of "file transmission" can be done is called a 'manager task'. Every task can be declared a 'manager'.

1.5. An illustrative session

In the EUMEL system, the process of producing a letter is as follows:

SV call SUPERVISOR



The command 'begin ("my first task")' which is to be initiated by CR creates a task having the name 'my first task' in the user branch, i.e. below PUBLIC. If this task were already available, you could have it displayed on the screen by entering 'continue ("my first task")'.

```
enter command:
edit ("invoice of 31.12.86")
```

The command 'edit ("file name")' opens a file in the task. If this file is new, you are prompted for confirming the spelling of the file name. If it is correct, confirm it by entering y.

In our example, the file already contains some text. Please enter an optional text and terminate the editing of this first file by actuating the keys ESC q (in sequence!).

```
.............. Invoice of 31.12.86 ..................  line  1
                                          G M D
                                          Sankt Augustin
                                          Schloss Birlinghoven

Ladies and Gentlemen,
```

In order to terminate the work in the task, enter ESC q (in sequence!).

After you have left the task, the EUMEL basic menu is displayed again. Start any further action again in this menu by entering SV. Especially for deactivating the device the task 'shutup' must be initiated after having entered SV (see also Appendix).

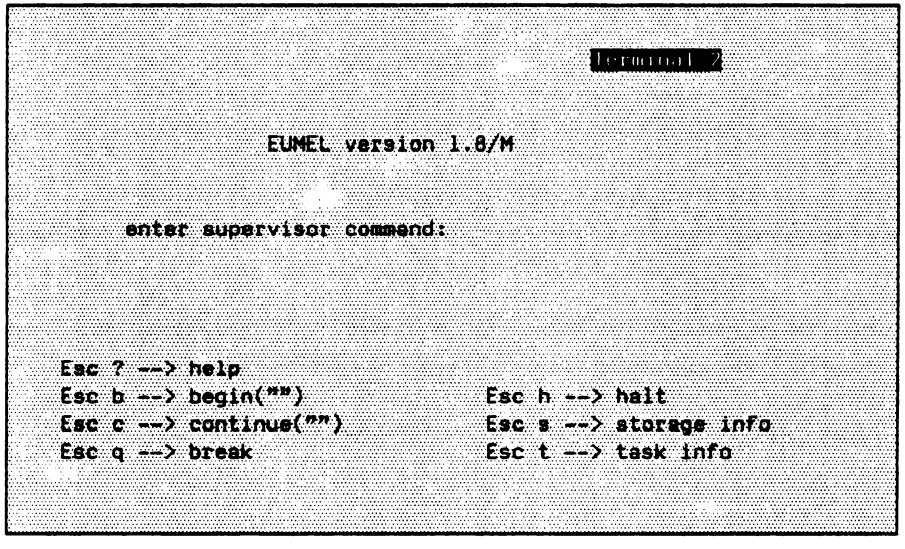
PART 2: The Supervisor

2.1. Control commmands

Every activity in the EUMEL system begins with calling the SUPERVISOR by actuating the key

SV

This keystroke connects your terminal to the computer. This procedure is also necessary when computer and terminal virtually form a unit.



The commands recorded on the lower information lines are now for selection. All these commands can be entered either by two keystrokes, i.e. ESC and identifier, or by typing their complete character strings. The commands are executed by means of CR.

The entry of a wrong character after ESC or of a wrong command is rejected. In this case, repeat the entry.

Function of the commands:

1. Control commmands

ESC b	begin ("taskname")	create task.
ESC c	continue ("taskname")	reconnect task.
ESC q	break	disconnect terminal.
ESC h	halt	discontinue program run.

2. Information commands (only Supervisor)

ESC ?	help	information.
ESC s	storage info	display used storage space.
ESC t	task info	display available tasks.

2.2. Creating a task

The command 'begin' creates a new task.

First, connect terminal and computer, then create a new task.

Connect terminal: SV

ESC b

The key combination 'ESC b' activates the insert mode and positions the cursor at the place where to enter the task name.

```
enter supervisor command:
begin ("  )
```

Enter file name:

```
enter supervisor command:
begin ("task name")
```

After having entered the name, actuate the CR key. This makes the Monitor of the new task reporting and you may enter any Monitor command (see Part 3).

If a task is created in the described way, it is automatically created (by the task SUPERVISOR) as a son of the task PUBLIC.

If a task is not to be created as a son of PUBLIC, but as son of another task, enter the command 'begin' followed by two parameters. The new task is then created as a son of another manager task (see Part 3).

```
enter supervisor command:
begin ("task name","name of father task")
```

NOTE: The task which is specified as a father task must be a manager task, otherwise nothing happens at all (see chapter 3.1.2).

2.2.1. Reconnecting a task

The command 'continue' is used for reconnecting an available task.

For resuming the work in a task, connect the task to the terminal by using the command 'continue'. This procedure resembles the creation of a new task:

Connect terminal: SV

ESC c

The key combination 'ESC c' activates the insert mode and positions the cursor at a place suitable for entering the task name.

```
enter superviser command:
continue ("task name")
```

After this input, you will find the continued task in the state you left.

2.2.2. Disconnecting the terminal

The command 'break' is used for disconnecting the terminal from the computer.

For example, if you want to disconnect the terminal from the computer immediately after an information command (see Part 2.3 ff), enter the 'break' command. After 'storage info', however, you can continue only with SV further.

```
enter supervisor command:
break
```

After this input, the terminal is disconnected. Any new activity is to be initiated with SV.

2.2.3. Stopping a running program

The command 'halt' stops a program currently connected to the terminal.

This command is important in special error situations. If you want to abort a program, but the display does not allow any regular entries, enter first SV.

As soon as the Supervisor display appears, actuate the keys

ESC h (or type 'halt' and actuate 'CR').

```
enter supervisor command:
halt
```

After this input, the program connected to your terminal is aborted and you return to the Monitor level (see Part 3).

2.3. Information commands

The information commands are used for obtaining information about the system.

The following information commands can be sent directly to the Supervisor.

Connect terminal: SV

ESC s

or

```
enter supervisor command:
storage info
```

supplies information about the used storage space on the EUMEL backing storage.

The command:

```
enter supervisor command:
task info
```

provides information about the names of the tasks available in the EUMEL system and the structure of the task tree. Branches in the task tree are represented by indentions at the levels of the task tree.

All tasks printed in bold face in the task organization scheme (see Part 1) are also available on any multi-user system since they are necessary for the operation.

The tasks below PUBLIC are, if any at all, named after their "owner" or the work done in them.

2.4. Overview of Supervisor commands

This chapter represents all Supervisor and task commands in ELAN notation.

The Supervisor commands are, as any other commands in the EUMEL system, in accordance with the ELAN syntax (command names are written in lower cases, parameters are enclosed in parentheses, several parameters are separated by commas, TEXT parameters are enclosed in double quotes etc.).

The ELAN notation

This notation is used for precisely describing the constructs of the programming language ELAN. The partly rather informal formulation within the chapter is followed by a brief description of the construct belonging to the specific subject area at the end of each part.

Such a description is, for example, as follows:

```
PROC edit (TEXT CONST file name)
```

The names of procedures, parameters etc. written in lower cases are self-explaining, we hope, the terms written in upper cases are so-called keywords and have the following meaning:

OP Operator
An operator effects an elementary operation. Operators are always represented by upper cases or special characters.

Example: + (addition of two numbers)

PROC Procedure
Program which is callable under its name, where appropriate with parameters added. CR terminates the input and makes the program run.

Example: 'edit ("file name")'

CONST Constant
Invariable value.

VAR Variable
Variable value.

BOOL Truth value
Type which can assume only the values TRUE or FALSE.

TEXT Text
Type which may include every alphabetic, special, but also numeric character. A TEXT CONST is therefore a so-called character string:

"my file"
"$abc123(XYZ)"
"account of 30.09.86"

A file generated in the Editor consists exclusively of TEXTs. A text is enclosed in double quotes " ".

INT Integer

Integer. An INT CONST is therefore any integer. The notation: 'INT CONST lineno', means that the number of the desired line of the file is to be specified in this place, i.e. '25' or '999'.

REAL Real

Real number. A REAL CONST is a number with decimal point.

PROC sin (REAL CONST x) => sin (0.5)

TASK Task

A TASK CONST identifies an existing task by an internal task name.

THESAURUS

A THESAURUS is a list of names, e.g. a list of file names.

The following Supervisor commands are available:

begin

PROC begin (TEXT CONST task name)
Creates a new task as a son of PUBLIC.

PROC begin (TEXT CONST task name, father task name)
Creates a new task as a son of the 'father task name' task.

break

PROC break
Disconnects the terminal from the computer.

continue

PROC continue (TEXT CONST task name)
Connects an available task to the terminal of the user.

halt

PROC halt
Aborts the running programs of the task currently connected to the terminal. The task is of course not erased.

To put it in more exact terms:
The error 'halt from terminal' is induced. Normally, this aborts the program as in case of any other error. For more details, please refer to the System Manual under error handling.

storageinfo

PROC storage info
Provides information about the backing storage.

taskinfo

PROC task info
Provides information about all task names in the system and indicates the father – son relations by indentions.

PART 3: The Monitor

3.1. The Monitor

> The Monitor receives the commands the user enters within his/her task. The present chapter describes the commands used for word processing.

The Monitor is the recepient of the commands within a EUMEL task. Each task and, therefore, each active user of a EUMEL system has a specific Monitor. Please remember in this context:

The SUPERVISOR is that task which supervises all other tasks of the EUMEL system.

The Monitor is the recipient of the commands in your task. Each task has a Monitor.

The Monitor in your task directly reports by the line:

```
enter command :
```

The commands at the Monitor level you may enter at this place can be divided into the following groups:

Information commands

storage info	display used storage space
task info	display tasks available in the system
task status	display status of task

Commands for task control

break	disconnect task
end	erase task
global manager	turn a task into a manager task, i.e. enable the creation of son tasks

Commands for file editing

copy ("file name","copy")	copy file
edit ("file name")	call Editor
forget ("file name")	erase file
list	list files
rename ("file name","new")	rename file

Transport of files

fetch ("file name")	fetch file from father task
erase ("file name")	erase file in father task
save ("file name")	send file to father task

Archive commands

archive ("name")	reserve archive drive
fetch ("file name",archive)	fetch file from archive
save ("file name",archive)	write file to archive
list (archive)	list contents of archive
check ("file name",archive)	check archive for readability
clear (archive)	erase/rename archive
format (archive)	prepare archive diskette for use

Word processing

list fonts	specify the set font types
fonttable	set desired font table
lineform ("file name")	format line by line
autoform ("file name")	automatic line formatting
pageform ("file name")	format page by page
autopageform ("file name")	automatic page formatting
index ("file name.p")	generate index
outline ("file name")	generate outline or abridged version of a text
print ("file name")	print file

Protection by password

task password ("secret")	define password for available task
begin password ("secret")	define password for new task
family password ("secret")	define password for several tasks
enter password ("secret")	define password for file

3.1.1. Information commands

The information commands supply information about your own task or the overall system.

The information commands 'ESC s' and 'ESC t' are known from the earlier Part 2. The abbreviating style is not predefined at Monitor level.

```
enter command :
storage info
```

provides information about the storage space used on the EUMEL backing storage.

```
enter command :
task info
```

provides information about the names of the tasks available in the EUMEL system and the structure of the task tree.

At Monitor level, additional information is obtainable by specifying a number between 1 and 3.

```
enter command :
task info (2)
```

supplies the following:

```
26.11.86   10:10                    CPU       PRIO   CHAN   STATUS
SUPERVISOR.....................  0001:08:50      0      -     wait
   -...........................  0000:00:08      0      2     i/o
   -...........................  0000:01:45      0      -     wait
   SYSUR.......................  0000:01:48      0      -     wait
      configurator.............  0000:00:43      0      -     wait
      OPERATOR.................  0000:00:03      0      -     i/o
         shutup dialog.........  0000:03:08      0      -     i/o
      ARCHIVE..................  0000:03:03      0     31     wait
UR.............................  0000:00:43      0      -     wait
   PUBLIC......................  0000:01:26      0      -     i/o
      agfa.....................  0000:00:11      0      -     i/o
      harry....................  0000:06:00      0      1     -busy-
```

'task info (1)' has the same effect as the command without parameter specification, '(2)' supplies in addition the used CPU time (= pure computing time), the priority, the channel and the task status for every task of the system. Apart from this information, '(3)' also delivers the storage space used by each task. The execution of task info (3) is very time – consuming!

In particular, for having a look at the storage space used by your own task, but also at the other information mentioned above, use the command:

```
      enter command :
      task status

24.12.86 18:30        TASK: bobby

Storage : 1000K
CPU time: 0000.01:11
Status  : -busy-, (prio 0), channel 1

      enter command:
```

3.2. Task control

3.2.1. Disconnecting a task

The command 'break' disconnects a task from the the terminal.

Entering the command 'break' at Monitor level disconnects the task from the terminal. The command effects no other changes.

```
enter command :
break
```

Instead of entering 'break', you may also use the key combination

3.2.2. Creating a manager task

A task can be declared a manager, i.e. a communication partner of other tasks. In particular, between manager tasks and other tasks being in a father – son relation, a simple file transfer is feasible (see p. 19).

A user task is usually created as a son of the task PUBLIC. You may however wish to introduce your own task hierarchy and to turn an available task into the father of one or several tasks to be created in the future in order to obtain thus a file hierarchy with the required operations. To do so, the respective task is declared a 'manager':

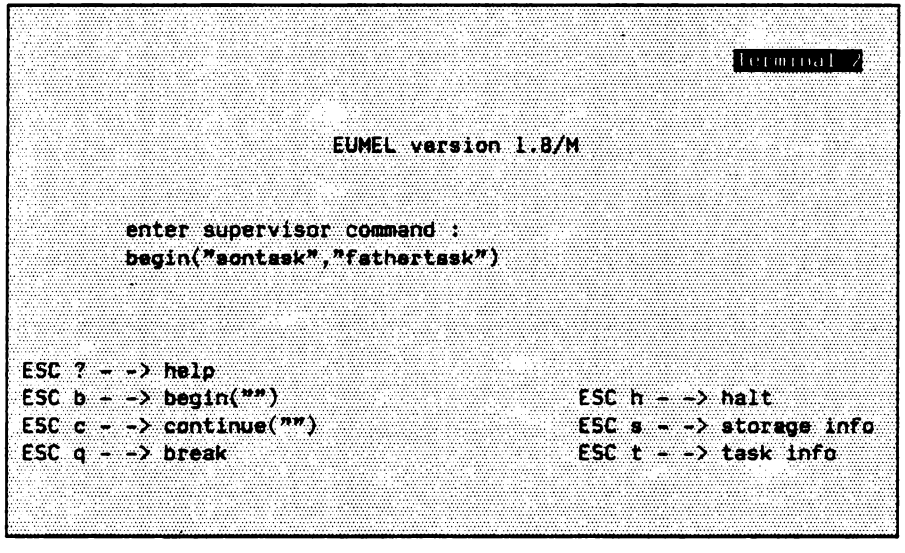
```
enter command:
global manager
```

With the global manager command, a 'break' command is implicitly entered which enables you to enter a Supervisor command after that command. Whenever you reconnect this (for the moment potential) father task ('continue' command), the task will not report as usual with 'enter command:', but with:

```
maintenance:
```

to indicate that it is a manager task.

For creating a son task below a manager task, not only the desired name but also the name of the father task is specified. [1)]



In this son task, files can then be fetched from and sent to the father task by entering simple commands.

[1)] If no father task is specified, the new task is a son of the manager task PUBLIC.

If a task is to be accessible by any other task, not only by the son task, this task must be declared a free manager.

```
enter command :
free global manager
```

Such a task can be accessed by any other task, otherwise apply what has been said of ordinary manager tasks.

3.2.3. Erasing a task

A task (and all data it contains) is erased by using the 'end' command.

Normal user tasks are created for a specific purpose or subject. If the work connected with such a task is done, the task should be erased after having saved all important data on a diskette.

```
enter command :
end
```

As for every command in the EUMEL system, the Monitor asks for confirmation:

```
enter command :
end
erase "task name" ? (y/n) ?
```

Only in case of a positive answer, the task is erased and the files are irretrievably lost. The following has the effect of a positive answer: y Y

n or N suppresses the offered action, any other input is rejected.

NOTE: If a manager task is erased, all son, grandson etc. tasks are also erased without asking for additional confirmation if the first request for confirmation is answered affirmatively.

3.3. File editing

3.3.1. Creating a file

The Editor call 'edit' creates a new file or displays an available file for editing.

A file contains logically interconnected texts and is uniquely identified by its name.

The EUMEL system stores written texts until they are erased by the user. In general, not only one (long) text or a program text is written, but several and different texts. In order to discriminate the different texts, they are provided with a name to be chosen freely. Examples of names:

```
"letter of 1.12.86"
"1st chapter of my book "
```

A collection of characters (i.e. usually our written texts) that has been provided with a name is called a **file**. Therefore, the Editor creates a file when we write a text. A file may comprise up to 4,000 lines of a length of up to 32,000 characters each.

Creating the first file in your task:

As a first step you should devise an appropriate file name. In practice, the EUMEL system does not restrict length or form of the file name. You should therefore accustom yourself to name your files in such a way that you will remember the contents of the files even after a week or longer.

A good name for the first file would be, for example: "my first file" or "test harry of 1.12.86". In the following, we will use only "file name" etc. Please replace this "file name" by the name you have chosen.

```
enter command :
edit ("file name")
```

```
enter command :
edit ("file name")

create "file name" (y/n)?
```

If you press 'y', a new file is created under the name entered by you. The file is still empty:

```
..... file name ..... line 1
```

Refer to Part 4 for further details on using the Editor facilities. Please write only some words into the file allowing its contents to be recognized. Use the keyboard just as that of a typewriter.

```
..... file name ..... line 1
Contents of the first file. 1234567890
```

The file should be closed here again.

To do so, please actuate ESC q.

It does not matter where the cursor is positioned.

Please repeat the creation of a file with a second file "other file name". Please write some characters into the file, too.

```
enter command :
edit ("other file name")
```

Enter the following, for example:

```
                    other file name                    line 1
Please keep any key pressedddddddddddddddddddddddddddd
```

Terminate work using again ESC q.

3.3.2. Listing file names

The command 'list' is used for listing the names of the files contained in a task.

Each file name is preceded by the date of the last update.

```
enter command :
list
```

```
                              list                              line 1
01.08.86 "file name"
01.08.86 "other file name"
```

This list of file names is a EUMEL file, too (though a write-protected one), output is therefore terminated as usual by entering the command

ESC q

3.3.3. Duplicating a file

The command 'copy' duplicates an available file.

An available file can be duplicated by entering the command:

```
enter command :
copy ("file name","copy name")
```

This command creates a copy of the file "file name" under the name "copy name", the contents of the two files are identical at first. Please check the correctness of this assertion by comparing the contents of the copied file with that of the original file "file name":

```
enter command :
edit ("copy name")
```

3.3.4. Changing a file name

The command 'rename' changes the name of the file.

If you do not like the name of a file, you can change it.

```
enter command :
rename ("file name","new file name")
```

3.3.5. Erasing a file

The command 'forget' erases a file.

A file is erased by entering the command:

```
enter command :
forget ("new file name")
```

To be sure, EUMEL asks, however, for confirmation before executing the command:

```
enter command :
forget ("new file name")

erase "file name" ?   (y/n)
```

The following has the effect of an positive answer: y Y

n or N suppresses the offered action, any other input is rejected.

3.3.6. Transferring files

Files can be sent to and fetched from the father task.

The convention that files are local to a task, i.e. that they are accessible only within this task, is often too restrictive. Thus it may be expedient to collect the essential results from several workstations (= tasks) in a central place or to store long term results from tasks which were created only for a short time to accomplish a special task. For this purpose, a user task is declared a manager (see p. 8) and sons of this task are created.

3.3.7. Sending a file to the father task

The command 'save' sends the copy of a file to the father task.

```
enter command :
save ("file name")
```

If a file is sent to the father task, a copy of the original file is created under the name 'file name' in the father task. After that, these (for the moment identical) files are independent of each other. Any modification to a copy will have no effect on the other copy of identical name residing in another task. If a file of the name 'file name' is already available in the father task, be it by accident or by executing a 'save' operation, EUMEL will report as follows.

```
enter command :
save ("file name")

overwrite "file name" ? (y/n)
```

Only if you enter 'y', the file in the father task is overwritten by your own file.

3.3.8. Fetching a file from the father task

The command 'fetch' fetches the copy of a file from the father task.

In analogy to sending a file copy, you can get a copy from the father task and, where appropriate, overwrite your file of the same name, of course after the necessary confirmation.

```
enter command :
fetch ("file name")
```

3.3.9. Erasing a file in the father task

The command 'erase' erases a file in the father task.

If a file is to be erased in the father task, enter this command analogously to the 'forget' command in the son task:

```
enter command :
erase ("file name")
```

If the file is available in the father task, it is erased after the required confirmation.

```
enter command :
erase ("file name")
erase "file name" ? (y/n)

enter command :
```

Note: The task 'PUBLIC' is always a manager task. Since a user task is created as a son of 'PUBLIC', unless it has been created as a son of a special manager task, the commands 'fetch', 'save' and 'erase' refer to 'PUBLIC'.

3.4. The archive

The archive is used for storing files on diskettes (backup).

The archive manages long-term data storage in the EUMEL system. Use the archive for:

- having backup copies of important files outside the computer;
- storing currently not needed files outside a task (saving storage capacity);
- transferring files to other computers.

The archive is implemented in the EUMEL system by the task 'ARCHIVE' which manages the disk drive. The control by a single task makes archive handling hardly any different from the file operations already known. Just specify in the commands that the archive is to be addressed.

3.4.1. Archive commands

Archive operations always consist of reservation, write or read access and archive release after the completion of work. Every archive operation begins with the reservation command.

As a first step of archive use, reserve the archive, i.e. communicate to the management of your EUMEL system that the task ARCHIVE controlling the disk drive is to work for your task. While the archive is reserved for your task, no other task is able to use the archive drive.

For the reservation, you should know the name of the prepared or recorded diskette you intend to use (the diskette label should show this name) or you should define a name (and write it onto the label) for a new diskette to be used. Naming is not regulated. Only after having entered the reservation command:

```
enter command :
archive ("diskette name")
```

mount the diskette onto the drive in order to prevent another user who happens to have already reserved the archive from working on your file which has incidentally the same name.

The command:

```
save ("file name",archive)
```

writes a file to a diskette and the command:

```
fetch ("file name",archive)
```

fetches a file from a diskette. For obtaining the directory of a diskette, enter:

```
list (archive)
```

3.4.2. Using a new archive diskette

A new diskette is first to be prepared (formatted) for use.

Before its first use, an archive diskette is to be formatted, i.e. it is to be divided into tracks and sectors for positioning the read/write head of the diskette drive, thus allowing a recording of the diskette. Formatting is device-dependent. Usual formats are as follows:

40 tracks of 9 sectors each (360 K)
80 tracks of 9 sectors each (720 K).

After archive reservation, the first use of an archive diskette requires the command:

```
enter command :
format (archive)
```

The command:

```
save ("file name",archive)
```

writes a file to a diskette and the command:

```
fetch ("file name",archive)
```

fetches a file from a diskette. For obtaining the directory of a diskette, enter:

```
list (archive)
```

3.4.2. Using a new archive diskette

A new diskette is first to be prepared (formatted) for use.

Before its first use, an archive diskette is to be formatted, i.e. it is to be divided into tracks and sectors for positioning the read/write head of the diskette drive, thus allowing a recording of the diskette. Formatting is device-dependent. Usual formats are as follows:

40 tracks of 9 sectors each (360 K)
80 tracks of 9 sectors each (720 K).

After archive reservation, the first use of an archive diskette requires the command:

```
enter command :
format (archive)
```

Only after the following request for confirmation

```
enter command :
format (archive)

format archive "diskette name"? (y/n)
```

formatting is actually done and the diskette of the name "diskette name" is available for archive operations. Note: When formatting for the first time the "diskette name" is not displayed upon the request for confirmation.

NOTE: If an already recorded diskette is reformatted, all data on the diskette will be lost.

Some computers allow variable formatting. For generating a format different from standard format, specify also the coding of the required format if using such computers.

Example: For a 5 ff diskette device, you may select:
code 0: standard format
code 1: 40 tracks
code 2: 80 tracks
code 3: high density

'format (archive)' generates a standard format diskette just as 'format (0,archive)' does, 'format (3,archive)' generates a high density format.

3.4.3. Deleting/renaming a diskette

Used diskettes can be cleared and renamed.

If you want to erase the contents of a recorded archive diskette or change the name of a diskette, reserve the archive under the desired name: if you want to erase the contents, do that under the previous, available name. If you want to rename the diskette, reserve the archive under the new desired name. Please note that renaming erases all files written on a diskette. Then enter the command:

```
enter command :
clear (archive)
```

The execution of the command provides the mounted diskette with the name specified upon reservation. The directory written on the diskette is erased. Thus, the data possibly recorded on the diskette is no longer retrievable. The diskette is to be used like a newly formatted diskette. [1] Accordingly, reformatting is not required when reusing the diskette.

[1] The command 'format' implies 'clear'.

3.4.4. The directory of a diskette

The command 'list (archive)' lists the files wriiten on a diskette.

After archive reservation, a formatted diskette can be read or written on. For identifying the files to be fetched (= read) or for getting an idea of the space empty for recording, you first should have a look at the directory of the diskette.

```
enter command :
list (archive)
```

Example:

```
.......diskette name (100 K out of 720 K occupied).....
01.05.86 25 K "invoices april"
01.06.86 23 K "invoices may"
01.07.86 20 K "invoices june"
01.08.86 32 K "invoices july"
```

3.4.5. Reading and writing on a diskette

Reading and writing on a diskette is similar to the well-known operations of sending and fetching files.

Writing a file onto a diskette is identical with sending a file to the father task. The only difference lies in the fact that you have to specify the destination explicitly.

```
enter command :
save ("file name",archive)
```

Reading a file from a diskette is done accordingly:

```
enter command :
fetch ("file name",archive)
```

Similar to the communication between son task and father task, only file copies are fetched or recorded.

3.4.6. Exchanging the archive diskette

When mounting another archive diskette, reenter the command

```
enter command :
archive ("diskette name")
```

since archive reservation includes a simultaneous verification of diskette name and directory.

3.4.7. Terminating archive reservation

Release archive after use!

After having completed all desired operations on the archive, release the archive again.

```
enter command :
release (archive)
```

This command enables another task to use the facilities of the task 'ARCHIVE'. If you do not enter the above command and if you do not initiate another archive operation within five minutes, another task can reserve the archive by means of the command 'archive ("diskette name")'. This is to prevent a forgetful user from blocking the archive in a multi-user system.

3.4.8. Archive error messages

Archive operations may lead to error situations.

Attempting to fetch a file from the archive may lead to the following report by the archive system

```
read error (archive)
```

and to an abortion of the read operation. This will occur when the diskette is damaged or not readable for any other reason (e.g. un– adjusted disk drives). In such a case, the archive system records internally that the file cannot be read correctly. You may see this when entering 'list (archive)'. In the displayed list, the file name in question is marked by the addition 'with read error'. For reading this file, read it under its file name with the addition 'with read error'.

```
enter command :
fetch ("file name with read error")
```

In this case the file is fetched from the archive in spite of the read error (loss of information!).

To avoid such cases, the EUMEL system enables you to check archives or archive files after writing. This is done by means of the following command:

```
enter command :
check ("file name", archive)
```

This command displays possible read errors.

Further error messages of the archive:

* read impossible (archive)
  The archive diskette is not mounted or the door of the drive is not closed.
=> Mount diskette or close door.

* write impossible (archive)
  The diskette is write-protected.
=> If actually desired, remove write protection.

* archive not reserved
  The archive was not reserved.
=> Enter 'archive ("name")'.

* read error (archive)
  See 'cannot read'.

* write error (archive)
  The diskette cannot be recorded (anymore).
=> Use another diskette.

* not enough system storage
  No sufficient space is left in the system to load a file from the archive.
=> Erase files where appropriate.

* RERUN during archive access.
  The system was interrupted by switching off or reset during an archive operation.

* "file name" does not exist
  The file "file name" is not available in the archive.
=> Check archive with 'list(archive)'.

* archive's name is ...
  The mounted diskette has not the entered name.
=> Enter command 'archive' with correct name.

* archive is being used by task ...
  The archive was reserved by another user.
=> Wait your turn.

* "file name" cannot be saved (archive full)
  The file is too large for the mounted diskette.
=> Take another diskette for this file.

* archive inconsistent
  The mounted diskette does not have the structure of an archive diskette.
=> You have forgotten to enter 'format (archive)'.

* save/erase not permitted due to read error
  For archives with read error, write operations are prohibited since success cannot be guaranteed.

3.5. Commands for several files

Using the special operators 'ALL' and 'SOME', you can handle several files by means of one command.

It is often very useful and facilitates handling to enter a command to be executed for several files, such as upon archiving if you want to record all files updated during the day onto a diskette.

Since tasks have a name and since each task has a directory showing its files, it is possible to specify lists of files.

Internal task names

For addressing a task other than your own one or the father task, you should specify the 'internal task name'. This is for the following reason:

Due to the tree structure of the EUMEL system described in the introduction, commands without special parameters can only be entered for your own task ('edit'...) or the father task ('save'...). At the archiving stage, for example, it would therefore be required to send a file via the father of the father of the father ... to the son of the son ... to secure that the file be finally transported to the task 'ARCHIVE'. Instead of doing so, use a procedure 'archive' indicating the internal task name. This identifies the desired task internally without requiring any further action from you.

Important procedures supplying the internal task names are as follows:

myself	name of your own task
public	name of PUBLIC
father	name of father task[1)]
archive	name of ARCHIVE
printer	name of PRINTER

[1)] If no special manager was created, 'father' and 'public' would of course supply the same task, i.e. PUBLIC.

File directories

Each task has its own directory of files. Inspect the directory of your own task by entering the 'list' command. You will obtain the directory of another task by using, for example, the command 'list (archive)'. In this case, add the internal task name of the desired task for obtaining the directory.

There are special operators enabling you to use a directory together with other commands:

ALL	supplies the complete directory
SOME	provides the directory for selecting entries.

Together with an internal task name, one of the two operators follows a Monitor command as a parameter. The command will then have an effect on all files contained in the directory.

```
enter command :
fetch (ALL father)
```

All files of the father task are fetched in sequence, the well-known request for confirmation is displayed for files of identical names in order to get the permission to overwrite files.

For processing only some files of the directory, put the operator 'SOME' before the task name:

```
enter command :
fetch (SOME father)
```

First the directory of the file is provided. Erase all files not to be fetched to your task from the directory by

- overwriting the file name with blanks

  or:

- deleting them with HOP RUBOUT

  or:

- marking several lines. For doing so enter 'mark' at the beginning of the passage to be marked and extend it by means of the cursor keys as far as appropriate. After that you can delete these lines by means of

  ESC RUBOUT or

  ESC p

```
invoices april
invoices may
invoices june
invoices july
```

In the above-mentioned example, the files 'invoices june' and 'invoices july' are fetched from the archive following the command 'ESC RUBOUT' (=delete marked lines) and the command 'ESC q' (=terminate editing).

As a further facilitation, there is the procedure 'all' as abbreviation of 'ALL myself'.

Example: write all files to archive diskette.

```
enter command :
save (all,archive)
```

For more experienced users:

You may use the directories of several tasks for creating a new directory. This requires the following set operations on directories:

- \- difference
- \+ union
- / intersection

Example:

fetch (ALL father - ALL myself)

All files of the father task which are not yet contained in your own task are fetched.

3.6. Password protection

The EUMEL system allows files, individual tasks and complete branches of the task tree to be protected by passwords.

For protecting parts of your EUMEL system against unauthorized use, regulate the access by means of a password.

You may choose any text as a password. Please note, however, that a really effective protection will only be secured if you use neither a trivial password (e.g. your own first name) nor a word you cannot remember.[1)]

NOTE: There is a special password in the EUMEL system: "–". This password prevents the task it protects (e.g. UR) from being connected to a terminal. Therefore, it should never be specified for a normal manager task.

[1)] You should never forget passwords. A file protected by a password can only be reconnected if you know the password. If you have forgotten it, you can only erase the task.

Protecting a task by a password

The Monitor command 'task password' protects a task by a password that is to be specified before accessing the task by means of the 'continue' command.

```
enter command :
task password ("rosebud")
```

If a user attempts to connect the task protected by the password to his/her terminal using the 'continue' command, s/he first is prompted for the password. The task is connected only after the password has been specified.

When specifying the password, overwrite the dots displayed on the screen by the corresponding characters. By actuating ESC, the entered characters can be made readable.

```
enter supervisor command:
continue("task name")
          password:.......
```

The password protection secures that no unauthorized user can access the files and programs of the task directly. There are, however, two situations allowing an unauthorized access to files:

a) send files to the father task:
When files are transported to the father task ('save' command), other users can access the files (if they are allowed to access this task). Avoid this by specifying a file password. Please note that the password for files and the above password for tasks are independent of each other.

b) Files are fetched to a son task:
If the task is created as a father task ('global manager' command), the son task can be used for fetching files ('fetch' command) from the father task which is protected by a password. Therefore, you should prevent unauthorized users from creating sons of a task protected by a password. This can be done by using the command

```
maintenance :
begin password ("secret")
```

This command has the effect that you are prompted for the password when trying to create a son task. Please note that 'begin password' is independent of the task password and the file password.

The command 'family password' protects a complete branch of a EUMEL system against unauthorized access. To do so, the command

```
maintenance:
family password ("secret")
```

is entered (as normal Monitor command) in the father task of the branch to be protected. Thus, the password of all sons, grandchildren etc. of this task is set to

'secret' if they had no password before or if they have the same password as the calling task. A task in this branch having already its own password which is different from the 'family password' maintains this specific password.

Example: The command `'family password ("secret")'` is entered for 'PUBLIC'. In this case, the password of 'PUBLIC' and of all tasks of the user branch is set to 'secret'.

It is to be noted that only the current sons of the task are taken into consideration when the 'family password' is allocated. Sons created after the allocation of the 'family password' are not protected by this password.

Cancelling a password

In order to cancel a password, enter the password command with "" as a parameter:

```
maintenance:
begin password("")
```

This overwrites the password with an empty text.

File password

The password protection is somewhat more complicated for individual files of a manager task since, in this application, a distinction between write protection and read protection is made. Since, in this case, only a few files of the father task are to be protected against reading ('fetch'), writing ('save', 'erase') or both, this procedure needs the specification of file name, write password and read password.

```
maintenance:
enter password ("file name","write protection","read protection")
```

If the file is not to be protected against reading, specify "" as a read password (see cancelling a password). If writing and/or reading a file is to be completely prohibited [1], specify "–" as an appropriate password.

Before a son task can read or write a the file of a father task which is protected by a password, the 'enter password' command has to be entered before the 'fetch', 'save' or 'erase' command:

```
enter command : enter password ("write password/read password")
```

Only one password is therefore entered in the son task. If, as stated above, a '/' is contained in this password, the first part before the '/' is checked as a write password and the second part after the '/' as a read password. If no '/' is contained, the word is interpreted both as a write and as a read password.

1) Of course, you can edit the file in the manager task it belongs to as usual.

Example:
A file "texts" containing text models is created in the manager task. This file is to allow fetching (=reading) in some son tasks. The revised, i.e. updated, file is, however, not to be written back into the father task.

In the father task: `enter password ("texts","-","psw")`

In the son task: `enter password ("psw")`

If the password is entered incorrectly or not at all in a son task, the following message appears:

```
    enter command :
    fetch ("protected file")
ERROR: wrong password
```

Therefore, this file can be fetched only by users who know the read password. Overwriting the file is not possible since the write password cannot be specified (" – "!).

3.7. Monitor commands

ALL

THESAURUS OP ALL (TASK CONST task)
Supplies a thesaurus[1] containing all file names of the specified task (including the user task 'myself').

```
fetch (ALL father)
```

THESAURUS OP ALL (TEXT CONST file)
Supplies a thesaurus containing the file names available in 'file' (one name per line).

```
fetch(ALL "file list")
```

archive

PROC archive (TEXT CONST archivename)
Initiation of archive operations. 'archive name' is used for checking in all following archive operations to prevent other users from unauthorized use of the archive. Initiation is rejected if the archive is reserved by another user.

```
archive ("text diskette")
```

TASKS PROC archive
Supplies the internal task name for the use in file commands.

```
save ("file name", archive)
```

1) In this context, a thesaurus is a list of files (see also 2.4. The ELAN notation).

begin password

PROC begin password (TEXT CONST secret)
Prevents other users from unauthorized creation of a son task.

```
begin password("gmd")
```

break

PROC break
The task currently connected to the terminal is disconnected, thus becoming a background task.

brother

TASK PROC brother (TASK CONST task)
Supplies the internal task name of the specified "brother" task.

```
list(brother)
```

check

PROC check (TEXT CONST file name, TASK CONST task)
Checks whether the file 'file name' is readable on the archive.

```
check ("my file", archive)
```

PROC check (THESAURUS CONST t, TASK CONST task)
Checks whether the files contained in thesaurus 't' are readable on the archive.

```
check (ALL archive, archive)
```

clear

PROC clear (TASK CONST task)
Erases all files of the task 'ARCHIVE' and renames the diskette if a diskette name different from that used so far has been specified.

```
archive ("diskl"); clear (archive)
```

copy

PROC copy (TEXT CONST source, destination)
Copies the file 'source' into a new file of the name 'destination' in the user task.

```
copy ("file","new file")
```

Possible error messages:
"destination" already existing
"source" does not exist
too many files

edit

PROC edit

a) In the Monitor:
   Calls the Editor with the file name used last.

b) In the Editor:
   Prompts for the file name.

The following applies to any 'edit':
If 'edit' is called for the first time, the window takes up the whole screen. In case of a repeated 'edit' call, a window situated to the right below the current cursor position is opened.

PROC edit (TEXT CONST file name)
Calls the Editor with 'file name'.

```
edit("manual part3")
```

PROC edit (TEXT CONST file name, x, y, widthx, heighty)
As the above 'edit' call, but the window where 'file name' is editable can be defined. The parameters define an Editor window with the left upper corner on the screen coordinates 'x' and 'y' and a line width of 'widthx' and a number of lines of 'heighty'. If the Editor is called with 'edit ("file name")', edit ("file name", 1, 1, 79, 24) is implicitly called.

```
edit("note",5,5,44,12)
```

PROC edit (THESAURUS CONST t)
Edits all files contained in thesaurus 't' in sequence.

```
edit (ALL father)
```

end

PROC end
The task currently connected to the terminal is aborted and erased.

enter password

PROC enter password (TEXT CONST file, writepass, readpass)
The specified file is provided with a write password and a read password. The passwords are not taken into consideration in your own task. If the protection is to be complete, "-" is to be specified for the prohibited operation as a password.

```
enter password ("safe","data","system")
```

PROC enter password (TEXT CONST password)
Specifies write and read password for the exchange with the manager task. If two different passwords are specified for read and write, they are to be entered as one string separated by "/".

```
enter password ("read/write password")
```

erase

PROC erase (TEXT CONST file)
Erases a file of the name 'name' in the immediate father task.

```
erase("old file")
```

Possible error messages: "file" does not exist
wrong password

PROC erase (TEXT CONST name, TASK CONST manager)
Erases a file of the name 'name' in the task 'manager'.

```
erase ("file name", father)
```

PROC erase (THESAURUS CONST thesaurus)
Erases the files specified in 'thesaurus' in the father task.

```
erase (ALL myself)
(* erases all files in the father task which are available in
the user task *)
```

PROC erase (THESAURUS CONST thesaurus, TASK CONST manager)

```
erase (all,father)
(* erases all files in the father task which are available in
the user task *)
```

father

TASKS PROC father
Supplies the internal task name of the user task's father task.

```
list(father)
```

TASK PROC father (TASK CONST task)
Supplies the internal task name of 'task'.

```
save ("file name", father (father))
(* copies 'file name' to the "grandfather" *)
```

fetch

PROC fetch (TEXT CONST name)
Copies a file from the father task to the user task.

```
fetch ("backup copy")
```

Possible error messages:
"file" does not exist
wrong password
too many files

PROC fetch, (TEXT CONST name, TASK CONST manager)
Copies a file to the user task of 'manager'.

```
fetch ("file name",/"global")
```

PROC fetch (THESAURUS CONST thesaurus)
Fetches all files contained in the 'thesaurus' from the father task.

```
fetch (ALL)
```

PROC fetch, (THESAURUS CONST thesaurus, TASK CONST manager)
Fetches all files contained in the thesaurus from the 'manager' task.

```
fetch, (ALL/"global",/"global")
```

forget

PROC forget (TEXT CONST file)
Erases a file of the name 'name' in the user task.

```
forget ("old file")
```

Possible error messages: "file" does not exist

PROC forget (THESAURUS CONST thesaurus)
Erases the files contained in 'thesaurus' in the user task.

```
forget (SOME myself)
```

format

PROC format (THESAURUS CONST thes)
Formats diskettes and sets the name.

```
format(archive)
```

PROC format, (INT CONST type, THESAURUS CONST thes)
Formats diskettes in the non-standard format of the used device.

```
format(2,archive)
```

global manager

PROC global manager
By calling this procedure, the user task is turned into a file manager. This allows sons of this task to be created.

list

PROC list
Lists all files of the user task by name and date of last access via terminal.

PROC list (TASK CONST task)
Lists all files of the specified 'task' by name and date of last update via terminal.

```
list (father)
```

myself

TASK PROC myself
Supplies the internal task name of the user task.

```
save (ALL myself, father)
```

public

TASK PROC public
Supplies the internal task name of "PUBLIC".

```
fetch ("file name", public)
```

rename

PROC rename (TEXT CONST oldname,newname)
Renames a file from 'oldname' to 'newname'.

```
rename("old manual","new manual")
```

save

PROC save (TEXT CONST file name)
Sends the file 'file name' to the immediate father task.

```
save("new manual")
```

Possible error messages: "new manual" does not exist
too many files
wrong password

PROC save (TEXT CONST name, TASK CONST task)
Copies file of the name 'name' to the task 'task'.

```
save ("file name",/"global")
```

SOME

THESAURUS OP SOME (THESAURUS CONST thesaurus)
Provides the specified thesaurus for editing. Names not desired can be cancelled.

THESAURUS OP SOME (TASK CONST task)
Provides a thesaurus of 'task' for editing.

THESAURUS OP SOME (TEXT CONST file name)
Provides a 'thesaurus' composed of 'file name' for editing.

storage info

PROC storage info
Provides information about the used backing storage space.

task

TASK PROC task (TEXT CONST task name)
Supplies the internal task name of 'task name'.

```
save ("file name", task ("PUBLIC")) = save ("file name",
public)
```

task info

PROC task info
Provides information about all task names in the system and indicates father – son relations (by indentions).

PROC task info (INT CONST type)
Provides information about all tasks in the system. Using 'type', you can select the type of additional information. Currently 'type' may assume the following values:

type = 1: corresponds to 'task info' without parameter, i.e. it supplies only the task names indicating the father – son relations.

type = 2: supplies the task names. In addition, you obtain information concerning the consumed CPU time of the task, the priority, the channel the task is connected to and the actual task status. The following values are displayed:

0	– busy –	task is active.
1	i/o	task waits for termination of output or for input.
2	wait	task waits for transfer from another task.
4	busy	blocked task is ready, but blocked.
5	i/o	blocked task waits for I/O, but is blocked.
6	wait	blocked task waits for transfer, but is blocked. Note: The task is automatically deblocked upon arrival of a transfer.

type = 3: as 2 with additional information of used storage space. (Please note, procedure is time – consuming!).

```
task info(2)
```

task password

PROC task password (TEXT CONST secret)
Defines a password for the user task. The command 'task password' is a Monitor command. If a task is protected by a password, the Supervisor will prompt for the password after the entry of the 'continue' command. Only after entering the correct password, you are able to access the desired task. The password can be changed by a repeated call of 'task password', e.g. if it has to be changed in regular intervals to protect personal data.

There is no possibility of disclosing a password once defined. If you have forgotten the password, you can only erase the task.

If a "–" is entered as a password, it secures that the task in question can never be connected to a terminal by means of the 'continue' command. This is useful, e.g. for manager tasks.

```
task password ("my secret")
```

task status

PROC task status
Provides information about the status of your own task and also about
- name of the task, date and time;
- consumed CPU time;
- used storage space;
- channel the task is connected to;
- status of the task (computing etc.);
- priority.

PROC task status (TASK CONST t)
Provides the same information as above, but about the task of the internal task name 't'.

```
task status (father)
```

THESAURUS OP + (THESAURUS CONST left, right)
Union of 'left' and 'right'.

THESAURUS OP + (THESAURUS VAR thes, TEXT CONST name)
Includes TEXT 'name' in the thesaurus 'thes'.

```
save (SOME father + "invoice", archive)
```

THESAURUS OP - (THESAURUS CONST left, right)
Difference of 'left' and 'right'.

THESAURUS OP - (THESAURUS VAR thes, TEXT CONST name) Supplies a thesaurus from 'thes', but without the entry 'name'.

```
save (ALL myself - "invoice" archive)
```

THESAURUS OP / (THESAURUS CONST left, right)
Intersection of 'left' and 'right'.

```
save(ALL myself / ALL father, archive)
```

TASK OP / (TEXT CONST task name)
Supplies the internal task name from a task name. "/" can be used in all cases where an internal task name is required.

```
fetch ("file name",/"global")
```

PART 4: The Editor

4.0. Preface

You use the EUMEL Editor to write all your texts and data. It offers multiple possibilities for supporting writers or programmers in preparing, correcting, and formatting manuscripts or programs. When writing the greatest support is provided in the form of permanent access to information already written (as a result of the storage capacities of computers). In contrast to working with a typewriter you can insert, correct, delete, and reorganize texts (as often as you like) with the EUMEL Editor.

This means that writing texts with the EUMEL system is particularly advantageous and time-saving when texts have frequently to be changed, or if they are to be printed in a particularly elegant way. Furthermore, the Editor offers writing aids, e.g. automatic word wrapping at the end of a line, automatic indentation, "learning" of texts to mention but a few. In addition to this the Editor capabilities can be extended and therefore be adapted to special writing demands. But this will be explained in one of the following chapters.

The developers of the Editor placed particular emphasis on the ease of operation: you can start writing within a couple of minutes and you can follow what happens to the text directly on the screen. The writing and correcting of texts is supported by a few but very efficient function keys.

Some of the text formatting functions cannot be "seen" on the screen, e.g. proportional spacing, bold face etc. Such services can be requested by means of directives to the text cosmetics programs and the EUMEL printer. These directives have to be inserted in the text. See Part 5 ("text cosmetics").

4.1. Turning the Editor on and off

Here we describe how the Editor is turned on and off and how the Editor creates a file.

When in your task the following request

```
enter command :
```

appears on the screen, you type

```
edit ("file name")
```

and the EUMEL Editor is turned on. If the file does not exist, i.e. no text is stored in the system under the specified name, the system then inquires whether a new file under the given name should be created:

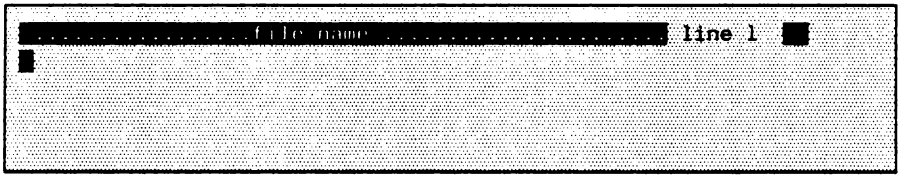
```
"file name" to be created (y/n)?
```

This function is a check for misspellings which can easily occur with similar file names. You can then decline to create the file, correct the file name, and enter the command again.

If you want to create a new file, you answer the request for confirmation with

y or Y

An empty Editor screen appears. The topmost line is the header line. You cannot write in it. It displays, however, some useful information: the file name, the current line number you are writing in, tab characters, rubin mode, learning mode, rest etc.



In this case you have created a new file. It does not contain any text yet. The header line, however, already displays the file name and the current line number. When there is a new file the screen will be empty under the header line. This serves as the "write field". The cursor is directly positioned below the header line. The cursor always indicates the current write position. Now you can start writing just like with a normal typewriter.

When calling a file in which you have already written a piece of text the Editor displays the text that was processed last and you can continue writing.

If you want to stop writing and to switch off the Editor, you press the following two keys

in sequence.

```
enter command:
```

appears and it signifies that you have left the Editor and are back to the Monitor level.

4.2. The most important Editor keys

4.2.1. The keyboard

There are a few keys on the keyboard which do not exist on a typewriter.

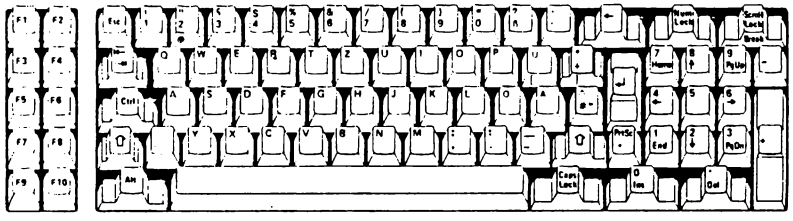
The keyboard layout of a EUMEL terminal corresponds to a large extent to that of a typewriter. One finds the letters 'a – z' and the digits '0 – 9' on keys. By pressing the 'SHIFT' key and another key simultaneously, you obtain upper case letters and a series of other characters, known as special characters. The "space bar" always produces a blank.

In practice, there are two different keyboards. First there is the EDP keyboard which is used for writing programs. You can recognize this by the fact that no umlauts ('ä','ö','ü') and no 'ß' are engraved on the keys. Instead, there are keys for square and curly brackets. If you want to write umlauts using such a keyboard, you must use a trick: by pressing 'ESC' and then activating another key (e.g. 'a','o','u'), you get the corresponding umlauts.

As a rule, the umlauts cannot be seen on the screen of this type of an EDP terminal, but rather they appear as 'a', 'o', 'u' etc. When a text is printed, however, they are displayed correctly.

The key assignment of the other keyboard corresponds to a large extent to that of a German typewriter and has keys for the umlauts and 'ß'. If the texts to be written are predominantly German, it is advisable to use such a terminal.

keyboard



In addition to these "simple" keys, there are function keys which are required when using the Editor (as well as other programs). The position of these keys depends on the type of terminal one uses. The function of these keys will be explained in the following sections.[1)]

1) It is possible that your keys may be not properly labelled. Consult your installation manual, it should clarify any difficulties concerning the counterparts. There may also be additional keys on your terminal, but they are of no real significance for the Editor as far as the standardized version is concerned.

The function keys of the EUMEL system

	positioning keys
SHIFT	case shift
CR	entry key, carriage return, end – of – line key
HOP	"booster key"
RUBOUT	delete key
RUBIN	insert key
TAB	tabulator key
MARK	mark key
ESC	command key
SV	supervisor key
STOP	stop key
PROCEED	proceed key

The function of the keys

SHIFT

Case shift

By pressing this key simultaneously with another key, one gets upper case letters instead of lower case letters and special characters instead of digits, e.g. instead of "9" the character ")" is displayed.

CTRL

Control key

In combination with other keys this one helps to select special system functions. For EUMEL the following three key combinations are important (but the keys have to be pressed simultaneously):

CTRL a	Stops a program or screen display.
CTRL b	Analogue to the 'SV' key (with any computer).
CTRL c	Proceeds the program or screen display.

Entry key / Carriage return / end – of – line key

This key is pressed in the Editor to mark the end of a paragraph. The continuous text input is interrupted by it and the cursor is positioned at the beginning of the following line. Indents are kept automatically. A paragraph mark is inversely displayed at the right – hand margin of the screen.

The 'CR' key is often marked by an arrow bending to the left. In command mode (i.e. when 'enter command :' is displayed) a given command is executed by pressing this key.

The use of this key outside the Editor is described in the corresponding application, e.g. accepting the suggested break point for hyphenation.

Positioning keys

Moving the cursor for one character/line in the respective direction.

"Booster key", serves as a "prefix" key

In combination with other function keys their effect is boosted (cf. p. 4 – 27).

Example:

HOP ↓

If the cursor is not placed at the bottom of the screen, then it is skipped to the last line. If it is in the last line of the screen, the screen display is turned by "a page".

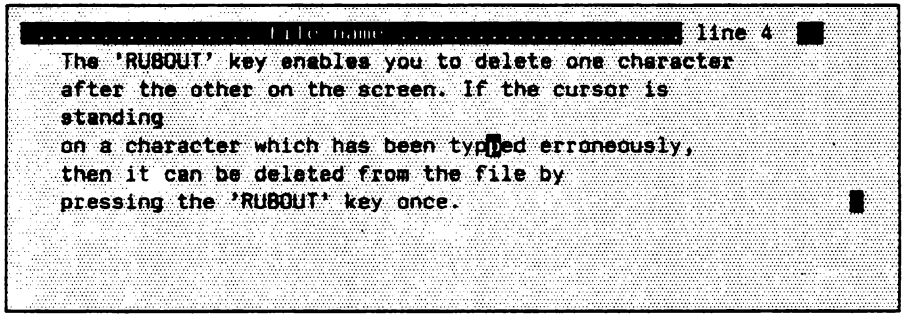
The functions of 'RUBIN'/'RUBOUT' are also boosted when used in combination with the 'HOP' key (cf. p. 4 – 28)

RUBOUT

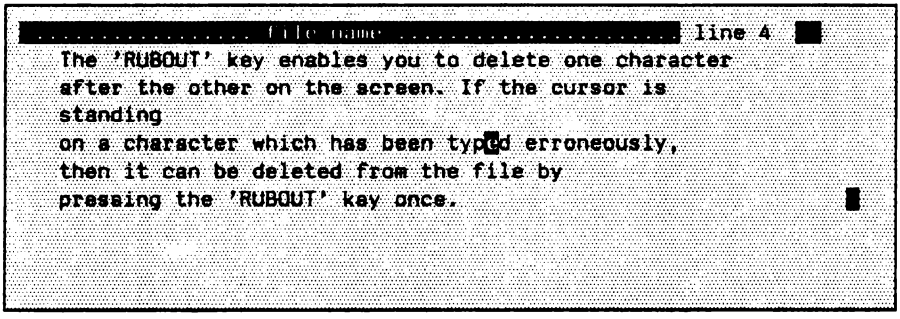
Delete key

The character on which the cursor is standing is deleted. If the cursor is behind the last character of a line, the last character of this line is always deleted.

Example:



After having pressed the RUBOUT key:



RUBIN

Turning the insert mode on or off.

Pressing this key activates the insert mode. This is indicated by the word 'RUBIN' in the left third of the header line. The insertion takes place in front of the character on which the cursor is standing. When pressing the 'RUBIN' key again, the insert mode is switched off.

Example:

```
.....file name..... line 4
By pressing the 'RUBIN' key the insert mode is switched
on.
This is indicated by the word
'RUBIN' in the left third of the header line. The
insertion takes place in front of the character on which
the cursor is
standing. By pressing the 'RUBIN' key again, the
insertion
mode is switched off.
```

Having pressed the **RUBIN** key and inserted the word "now":

```
.....RUBIN.....file name..... line 4
By pressing the 'RUBIN' key the insert mode is switched
on.
This is indicated by the word
'RUBIN' in the left third of the header line. Now The
insertion takes place in front of the character on
which the cursor is
standing. By pressing the 'RUBIN' key again, the
insertion
mode is switched off.
```

TAB

Tabulator key

Press the 'TAB' key to skip from the left screen margin to the beginning of the text in a line or a column. Pressing the 'TAB' key again makes the cursor skip to the next pre – set tabulator position. The setting of a tabulator can be seen from the tabulator marks ("carets") in the header line.

If there are no tabulator marks, the two margins, left and right screen margin, are taken as *pre – set* tabulator marks.

MARK

Turning the marking function on or off.

By pressing this key you switch into a special mark mode. Everything you are writing now or that you are marking by moving the cursor in direction to the end of the file is available for processing. For better identification the marked area is displayed inversely.

If the cursor is moved in one direction, the whole piece of text will be marked, starting from the moment you switch on the marker and the current cursor position. Moving the cursor backwards means a reduction of the marked area.

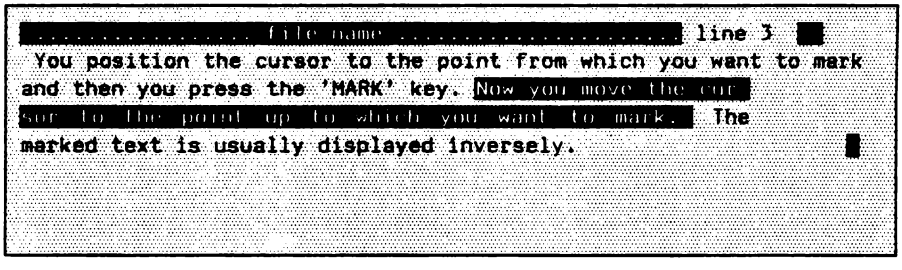
Such a marked area, for example, can now be duplicated, shifted, deleted, searched or further processed. (see p. 4 – 36 ff).

Pressing the 'MARK' key again turns the marker off.

Example:

You want to mark a certain area of text in order to move it somewhere else (possibly in order to delete it there later):

You move the cursor to the beginning of the text, turn on the marking function by pressing the 'MARK' key and move the cursor with the help of the positioning keys to the end of the area that is to be marked.



The area can now be processed further by other commands (see 'ESC' key and command processing, p. 4 – 35)

ESC

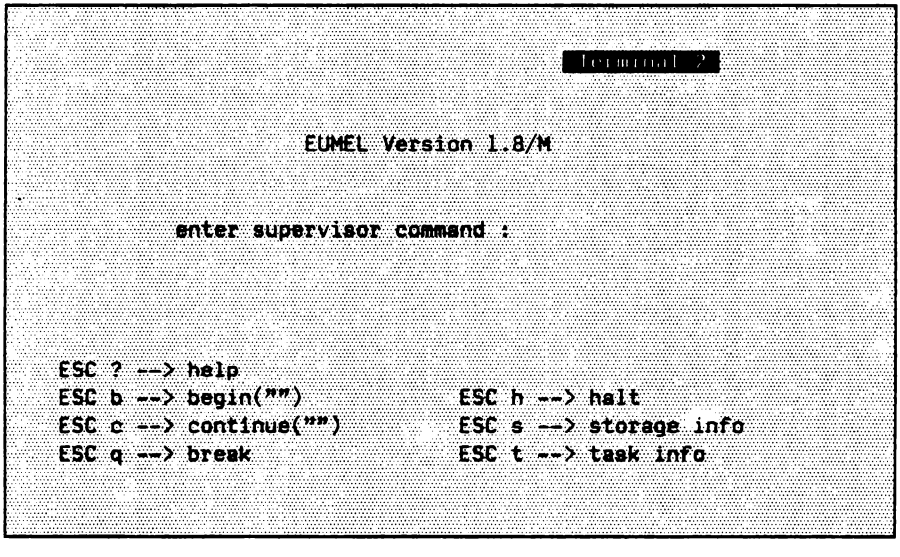
Command key

The 'ESC' key in combination with a following key enables you to select predefined actions. There are actions programmed and you can add further ones. (cf. p. 4 – 35)

SV

'supervisor' key in multi – user operation

By pressing this key in the Editor, you interrupt your Editor work and receive the message



If you want to continue in the Editor or if you pressed the 'SV' key by mistake, then you enter the command

```
enter supervisor command :
continue ("office")
```

(If the task you were working in, was actually called "office".)

In order to see the text that is being edited fully on the screen, you press the

ESC b keys.

You are back at the point where you left the text with the 'SV' key and you can continue working as usual.

NOTE: The 'SV' key can be implemented by two keys instead of one (often 'CTRL b'), depending on the unit you are using. Have a look at your keyboard.

STOP

Stops an output (this function is often implemented as 'CTRL a').

If you pressed this key by mistake, in which case the Editor does not 'react', press the 'PROCEED' key (often implemented as 'CTRL c').

PROCEED

Continues the interrrupted output.

The output terminated by the 'STOP' key can be continued by pressing the 'PROCEED' key.

NOTE: The 'STOP' key only interrupts the display on the screen. Characters that have been entered while 'STOP' was working are stored and read out after the 'PROCEED' key has been pressed.

4.2.2. Storing texts

This section explains the term 'file' and how different texts can be distinguished from each other.

The EUMEL system stores written texts until they are deleted by the user. As a rule, not just one (long) text or program is written, but rather several different ones. In order to be able to differentiate between them we give each of them a name which can be chosen freely. Examples of names are:

```
"letter dated from the 1st December, 1986"
"1st chapter of my book"
```

A collection of characters (usually the texts we have written) which have been given a name is called a **file**. The Editor therefore creates a file when we write a text. A file can contain up to 4,000 lines and each line can hold up to 32,000 characters. Together the number of lines and the characters per line cannot exceed 1,000,000 characters (1MB) at the moment.

4.2.3. Writing texts

Texts are written continuously. Paragraphs are marked by pressing the 'CR' key.

After this somewhat lengthy introduction we can now finally start writing. When a character is written, the cursor automatically moves to the right to the next character position. As a result of the automatic word wrapping function, words which would run over the end of a line are taken into the next line without hyphenation.[1)]

The 'CR' key (which means 'carriage return' on a typewriter) has only to be pressed when you want to end a line prematurely, i.e. for a paragraph or a blank line. The cursor is positioned at the start of the next line. At the same time a mark appears at the right margin of the previous line indicating the end of a paragraph.

Therefore the 'CR' key is particularly important for table lines and program texts because these lines should be kept separate. The key works behind the last character only.

The Editor is programmed for writing "normal" texts. With normal texts a word which would run over the end of the line is automatically taken to the beginning of the following line. This function is called "word wrapping".

---

1) Hyphenation should not be carried out "by hand". The separating characters are regarded as hyphens of a compound and are kept during reformatting which is undesirable. For this tedious task there is a program in the text cosmetics.

If word wrapping is not desired, e.g. when describing programs, you give the following command in the Monitor before calling the Editor

```
enter command :
word wrap (false)
```

The word wrapping function can be turned on again with the following command:

```
enter command :
word wrap (true)
```

Word wrapping is predefined in the Editor and it should only be switched off in exceptional circumstances.

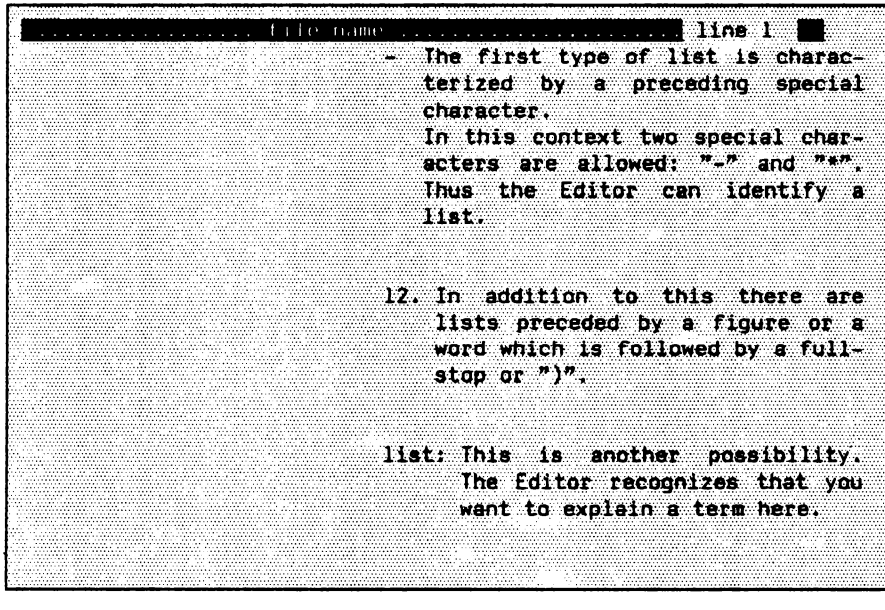
Apart from the header line a screen usually displays 23 lines in which text can be written. If the last line is "full" and if a new line is to be started, the screen contents automatically move up a line. This way you get a blank line which can be written on now etc. Don't worry: the lines which have disappeared have not been "lost". As the screen can only display a limited number of lines, the Editor can only show a part of a file.

4.2.3.1. Indents

Automatic indentation permits continuous writing because indents are kept automatically.

If a text is to be indented, the space bar is pressed accordingly. The indent written in this line is automatically kept in the subsequent lines until it is cancelled by the positioning keys.

Example of lists: Indents are automatically made without actively pressing the space bar.



When does the Editor recognize a list?

The following details should only be read by a user who is interested in this aspect.

If the automatic indentation function does not work, you should make sure that the following points that are essential for indentation are fulfilled:

1) The preceding line has got a paragraph mark.

   Important: Within a list item the automatic indentation function is switched off by the 'CR' key.

2) "*" or "–" and at least one blank are the first characters in the line.

3) "." or ")" and at least one blank after not more than seven characters are the first characters in the line.

4) ":" and at least one blank after not more than 19 characters are the first characters in the line.

4.2.4. Positioning within a text

In order to carry out corrections (overwriting, deleting or inserting), one must be able to move the cursor which indicates the current write position. With longer texts, it is also possible to position the cursor to lines which are not yet displayed on the screen. Therefore the Editor does not just show the end of file, but also any section which can be seen on the screen in the so-called window.

If corrections are necessary, you move the cursor to the position where the correction is to be made. The 'LEFT', 'RIGHT', 'UP' and 'DOWN' positioning keys are used for this. 'LEFT' and 'RIGHT' move the cursor within a line. If you press the 'RIGHT' key at the end of line, the cursor is moved to the beginning of the following line.

A line change can be performed more easily using the 'UP' and 'DOWN' keys. The 'UP' key moves the cursor one line up, the 'DOWN' key one line down.

What happens when you reach the upper or bottom edge of the screen and you continue positioning? In this case, the text is shifted up or down line by line and the line we want appears, others "disappear" over the other edge.

As a result we see that one can use the positioning keys to slide the screen over the file like a window. The text itself can be thought of as being written on a long band. The number of the line in which the cursor is positioned is always shown in the header line.

You should avoid to let the cursor run over the end of a text. This way you prolong your text with blank lines which you do not fill up when going on writing but you shift them in front of you.

Within a line it is somewhat different: If we position to the right in a line which is wider than the screen, the window is not shifted but the line is 'scrolled' (cf. shifting the whole window by the 'margin' command, p. 4 – 59)

4.2.5. Corrections within a text

Simple corrections can be made by overwriting, deleting and inserting characters.

RUBOUT

The simplest way of making corrections is overwriting. If, for example, a character is to be replaced by another, position the cursor exactly onto it and type in the correct character. This can also be done with several characters in sequence.

Corrections can be made while writing by deleting the last character written with the 'RUBOUT' key. Spelling mistakes are, however, often only noticed later, which means that these mistakes cannot be corrected so easily. In such cases, the cursor has to be moved to the position in the text at which the correction is to be made.

If you want to delete a character, again position the cursor onto this character and press the 'RUBOUT' key. The character disappears and the rest of the line closes together. If several characters are to be deleted, the 'RUBOUT' key has to be pressed a corresponding number of times.

If the cursor is behind the last character of the line, the last character is always deleted. You can, therefore, use this facility to delete a line "from behind". (cf. also p. 4 – 37)

RUBIN

Missing characters can just as easily be inserted. You position the cursor onto the character in front of which the missing character is to be inserted. Then press the 'RUBIN' key. The Editor goes into the insert mode which is indicated by 'RUBIN' in the header line. It inserts all the characters which are now typed (instead of overwriting). The part of the line to the right of the cursor moves a corresponding number of positions to the right.

What is important is that the Editor works exactly the same in the 'RUBIN' mode as it does in the normal mode (with the exception, of course, that the text is inserted instead of being written over).

No characters can be lost when the 'RUBIN' mode is switched on. Many users therefore leave the 'RUBIN' mode turned on to prevent texts from being written over unintentionally. Corrections are made by inserting the improvements and deleting the old text.

The insert mode is ended by pressing the 'RUBIN' key again. The 'RUBIN' key therefore acts like a switch for turning the insert mode on and off. However, you can only insert as many characters into a line until the last word of the line comes up against the end of the line. The last word is inserted at the beginning of the following line provided that there is sufficient space and that it is obviously not the last line of a paragraph. If this is not the case, a new line is automatically inserted for the word you have started. (see also p. 4 – 38)

4.2.5.1. Skipping and inserting/deleting lines

Using the positioning keys for moving the cursor over larger "distances" is somewhat tedious, which is equally true for extensive deleting or inserting. The 'HOP' "booster key" speeds up these operations in a simple way. The 'HOP' key can be used to move the window over the file not only one line at a time but also a whole window – length at a time. This is called 'paging'.

If the 'HOP' key is pressed before one of the previously explained function keys, it boosts its effect. The 'HOP' key is a "prefix" key: it is pressed before another key (and not simultaneously, like with the case shift 'SHIFT'). The skipping positioning will be explained first:

Positioning in lines with the cursor.

Skip to the right – hand end of line.

If the line is longer than the window is wide, the line is moved to the left by one window if the keys are pressed again.

Skip to the left margin (if necessary, paging sideways).

Skip to the first line of the screen.

Pressing this key combination again, positions the cursor (and the window in the file) up by one window ("paging").

Skip to the last line of the screen.

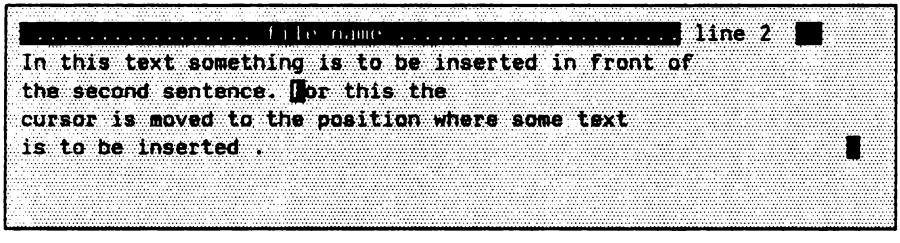
Paging is analogous to 'HOP' 'UP'.

Positions the window in such a way that the current line becomes the first in the window.

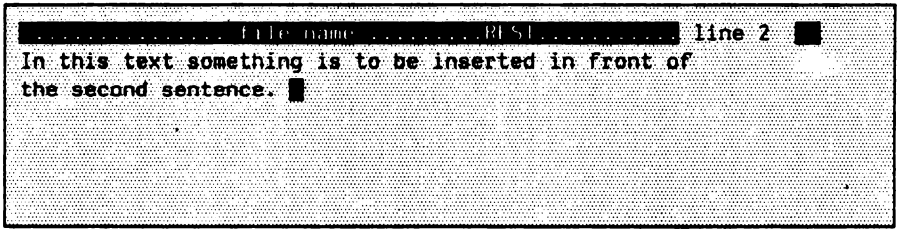
HOP RUBIN

Insertion of text passages. The 'HOP' key is used together with 'RUBIN' and 'RUBOUT' for "boosted" inserting and deleting.

The rest of the text behind the current cursor position "disappears". You can continue now as if you were entering a new text. The 'REST' flag in the header line reminds you that the rest of the text still exists. It appears on the screen again after renewed pressing of the 'HOP' 'RUBIN' keys (the 'REST' flag then disappears).



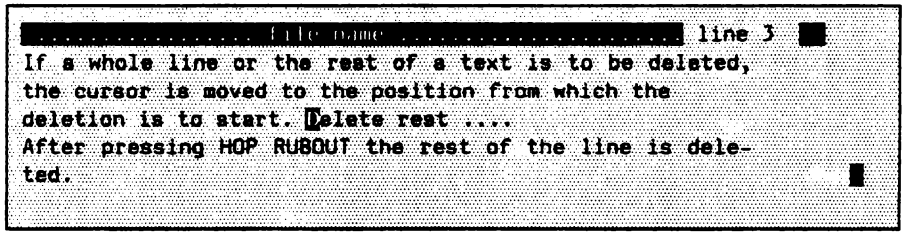
Having pressed the HOP and RUBIN keys the screen display looks as follows:



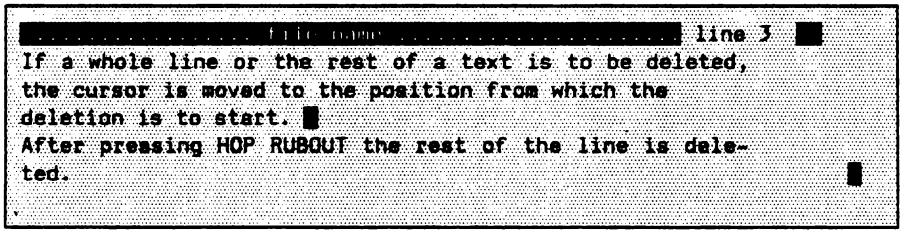
Now you can insert as much text as you like. Pressing 'HOP' and 'RUBIN' again, makes the rest of the text reappear starting in the line below the inserted section.

HOP RUBOUT

Deletes the line from the cursor position to the end of line.



Having pressed the HOP and RUBOUT the screen display looks like the following:



If the cursor is at the start of a line, the whole line is deleted and the gap is closed by the following lines moving up (when you press 'HOP' 'RUBOUT' again).

4.2.5.2. Splitting lines and re – makeup

'HOP' 'RUBIN' is pressed in sequence in order to insert larger text passages. This key sequence can be used to split a line or a larger text passage. 'HOP' 'RUBOUT' at the end of a line brings about a re – makeup.

HOP RUBIN

As already described, 'HOP' 'RUBIN' in a line makes the rest of a line to the right of the cursor and all lines below the current line disappear. 'REST' in the header line reminds us that a part of the file is not visible.

If 'HOP' 'RUBIN' is pressed immediately after 'HOP' 'RUBIN' again, the former rest of the line is displayed as an independent line. Thus one line is split into two.

HOP RUBOUT

The reverse of the above, i.e. combining two lines into one (so – called 're – makeup') is possible by means of 'HOP' 'RUBOUT' behind the last character of a line. The cursor can easily be positioned after the last character of a line.

The combined use of line splitting and re – makeup restores a line to its original condition. Example: 'HOP' 'RUBIN' is used to split a line, the rest of the line and the following lines disapppear from the screen. Pressing 'HOP' 'RUBIN' again, displays the former right – hand part of the line in the following line. The other lines have moved down a line. Since the cursor is still positioned at the right margin of the split line, you can recombine the original right part of the line using 'HOP' 'RUBOUT'.

4.2.6. The tabulator

A further important positioning aid within a line is the 'TAB' key. Among other things it is required for writing tables. Tabulator marks can be set or deleted as with a typewriter.

TAB

The tabulator fulfills an important function for fast positioning even if no marks are set. Pre-set tabulator marks mark the beginning of a line (indent, if present) and the position directly behind the last character of a line. Pressing the TAB key therefore makes the cursor skip to the next of these pre-set positions. This allows you to move the cursor quickly to the beginning or end of a line (and, for instance, to delete characters "from behind" at the end of a line or to continue writing there).

HOP TAB

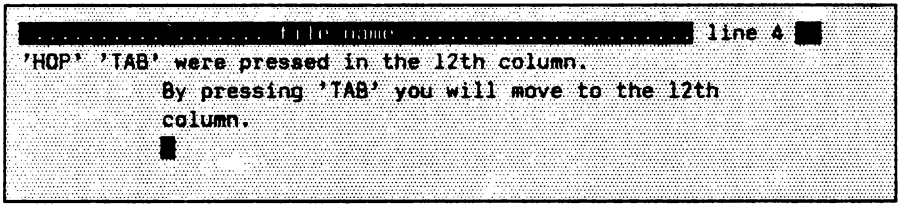
Let us set the tabulator. The tabulator is set by moving the cursor to the line position at which the mark is to be placed. Now press 'HOP' 'TAB'. The setting of the tabulator can be seen from a mark ("caret") in the header line (if it is in the window and if the current line has not been shifted sideways). Pressing the 'TAB' key now at any position within the line moves the cursor to the next tabulator mark (which is to the right of the cursor) or to one of the pre-set positions.

Set tabulator marks can be deleted by setting the position of the tabulator mark with the 'TAB' key and then pressing 'HOP' 'TAB'. The mark is then deleted and the caret disappears in the header line.

Tabulator marks leave no traces in the file, acting only as positioning aids. Marks which have been set with 'HOP' 'TAB' and which are positioned with 'TAB' have the same function as decimal tabulators when writing figures. (see p. 4 – 34)

Example:

The tabulator position is to be set on the 12th column for the beginning of a text. For this the cursor is positioned on the 12th column and the 'HOP' and 'TAB' keys are pressed in sequence. The "caret" appears in the 12th column of the header line and from now on this position can be directly reached by pressing the 'TAB' key.



If tabulator marks are set ('HOP' 'TAB'), the pre – set tabulator marks (beginning and end of line) become invalid. This is necessary e.g. when writing tables. We would, however, like to make use of the pre – set tabulator marks again when writing "normal" texts. The

ESC TAB

keys can be used to make the set tabulator marks (recognizable by the "caret" character in the header line) disappear. Then the pre – set tabulator marks become valid again. Repeated use of 'ESC' 'TAB' reactivates the set tabulator marks etc.

4.2.6.1. Writing number tables: decimal tabulator

When writing number tables, the numbers often have to appear right justified in the text. The Editor provides the decimal tabulator for this purpose.

The desired units place (i.e. the last place) is set for each column by using the tabulator. The cursor is advanced to the next tabulator mark with TAB. If you now want to enter numbers, they are not – as is normal – written from left to right, but rather are indented to the left. To be more precise: If there is a blank, a number, a "+", a "–" or a decimal point to the left of a number, it disappears when another number key is pressed and the new number sequence, which results, is written right justified at the tabulator position. Number columns can thus be written easily and right justified[1]:

```
   12          12345,78
    1              0,23
12345           1234,00
```

The system therefore has four useful automatic functions: apart from the automatic decimal tabulator, there is word wrapping, automatic identation and automatic line insertion when writing in the insert mode.

[1] If proportional spacing (type font whose characters show different widths) is used, you should have at least two blanks between the individual columns. Otherwise, due to the different character widths, the columns will not be printed right justified.

4.2.7. Learning mode in the Editor

Any sequences of keystrokes can be learned and assigned to keys. This is useful when the same keystrokes have to be repeated time after time, e.g. inserting into table lines or if the same texts have to be written frequently, such as sender, greetings etc.

ESC HOP

The learning mode is turned on by pressing the 'ESC' 'HOP' keys ('LEARN' appears as a reminder in the right part of the header line). All keystrokes (including keystrokes like 'CR') are now learned until the learning mode is switched off. You can, therefore, have the system learn several lines.

ESC HOP 'key' e.g. 'ESC' 'HOP' 'j'

Learning mode is terminated or turned off by pressing the three 'ESC' 'HOP' 'key' keys. The keystroke sequence that has been learned, which is also known as the learning sequence, is then assigned to the 'key' key.

ESC 'taste' e.g. 'ESC' 'j'

Subsequent use of the 'ESC' 'key' key sequence causes the text that has been learned to be written at any desired position in the file.

Example:

A typist has to type the words: 'Data Processing Company' 50 times a day. S/he gets the Editor to learn these words with

```
'ESC' 'HOP' Data Processing Company 'ESC' 'HOP' 'm'
```

The words are now on the 'm' key. If 'm' is pressed, an 'm' appears on the screen. The above written words appear with 'ESC' 'm'. 'ESC' is, therefore, necessary to distinguish the normal 'm' from the learning sequence.

Which keys can be reserved for learning? All keys, except for

- those used by the system, such as 'SV' and 'CTRL';
- keys that are pre-assigned by the Editor (depending on the application), such as the 'q' , 'ESC' or 'HOP' keys;
- keys defined by programming.

Practical tips: You should not overload the keyboard with learning sequences because you cannot remember a lot of keys. It is much better to reserve just a few keys and to keep the others for current tasks.

The use of learning sequences is particularly useful when writing directives for the text cosmetics programs. Directives such as 'turn on underlining', type font directives etc. are suitable for key assignment.

Don't worry when you happen to make a typing error when in learning mode: you can correct it immediately (e.g. with the 'RUBOUT' key). Such keystrokes are also learned, but they are of no importance when using the learning sequence.

By the 'ESC' 'HOP' 'HOP' command everything that has been learned is forgotten and the learning mode is switched off.

4.2.8. Editing text sections by marking them

It is often necessary to delete or shift several lines or entire text passages. The 'MARK' key, with which you can mark texts, is useful in this respect. The texts marked in this manner can then be edited, as a whole, in various ways.

MARK

Pressing the 'MARK' key turns the marker on and – by pressing it again – it is turned off. The beginning of the marking is "recorded" and you can now move the end of the marking towards the end of file by using the positioning keys and the 'HOP' key, and while doing so any characters in between are marked (usually displayed inversely).

ESC RUBOUT

A text marked in this way can be deleted by 'ESC' 'RUBOUT'. Marking and deleting with 'ESC' 'RUBOUT' is a convenient and safe method of deleting since you can see exactly what is being deleted.

ESC RUBIN

The deleted section, however, is not deleted completely, but rather it can be inserted again at another (or at the same) position in the text by means of ESC RUBIN. The "carefully" deleted text goes into an intermediate memory and can be called again, if necessary, by means of 'ESC' 'RUBIN'. If you carefully delete again, the last text of the intermediate memory is overwritten. In the intermediate memory there is only space for one text. This is a reliable, fast and easy method for moving a text passage of any length to another position. Furthermore, erroneous deleting operations can be readily corrected because the text can be reproduced easily with 'ESC' 'RUBIN'.

It is also possible to write when the marker is on. Marked writing is a particularly cautious method of text production because the text being inserted only really exists when the marker ('MARK') is turned off. It can be deleted again ('ESC' 'RUBOUT') and moved to another position ('ESC' 'RUBIN'). When using marked writing, 'RUBOUT' always acts on the character that is in front of the cursor.

Note: Positioning is only possible within the marked section.

4.2.9. The window Editor

It is often necessary to work with several files simultaneously, e.g. when something has to be copied from one file into another, when the text cosmetics programs or a compiler detect errors or when you want to look up something in another file. For this purpose the Editor provides the possibility to edit two (or more) files at the same time.

The Editor enables the user to look at the text that is to be processed like looking through a window. In this context it is quite normal that during the editing of a text one has the wish to see additional texts simultaneously. This can be necessary in order to compare something, to discover errors or to transfer text passages from one window to another one.

Pressing the following keys in the Editor "opens" a new Editor window:

Pressing 'ESC' 'e' approximately in the middle of the screen displays the window on the new file in the lower half and the "old" file in the upper half of the screen. First the file name is requested. After entering it and pressing the 'CR' key a window is opened onto another file. The upper left corner of the window is placed at the current cursor position. The cursor should not be too close to the right or to the lower edge because otherwise the window would be too small. You can work in this window in exactly the same way as in the "normal" Editor.

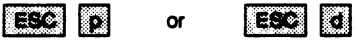
The key sequence

can be used to change from one window to the next (on a cyclic basis). There is a hierarchy between the windows in the sequence in which they have been created. If

is pressed in a window, it disappears and all the ones that are nested in it, too. Once again you are back in the higher window.

We described before that with the help of 'ESC' 'RUBOUT' and 'ESC' 'RUBIN' texts can be shifted and deleted. From one file to another in the window Editor this happens in the following way:

The command



can be used to write a marked text into a temporary file (an intermediate memory); the command 'ESC' 'p' removes a marked text out of the original file and writes it into an intermediate memory. In contrast to that a text is copied with 'ESC' 'd'. The command

is used to insert the text into another (or the same) file. In contrast to 'ESC' 'RUBIN' the contemporary file is not emptied then.

The 'ESC' 'd' and 'ESC' 'g' functions perform the same as the 'PUT" "' and 'GET" "' commands but more quickly.

4.2.10. The most important pre – assigned keys

Learning sequences and commands (i.e. ELAN programs) can be assigned to keys. Since a few functions are used more often, there is a standard pre – assignment of certain keys.

ESC q	leave the Editor or the nested windows.
ESC e	create another Editor window.
ESC n	"open" note book.
ESC v	enlarge the file window onto the whole screen or reconstruct screen (leave nested window).
ESC w	change file in the window Editor.
ESC f	repeat execution of the last command.
ESC b	the window is moved to the left edge of the current (if appropriate, shifted) line.
ESC →	go to start of next word.
ESC ←	go to start of last word.
ESC 1	go to start of file.
ESC 9	go to end of file.

Learning operations

ESC HOP	learning mode is turned on.
ESC HOP taste	learning mode is turned off and learning sequence is assigned to 'key'.
ESC HOP HOP	forget learning sequence. It is a condition that the learning sequence is deleted in the same task in which it has been learned.

Operations with markings

ESC RUBOUT	"cautiously" delete marked text.
ESC RUBIN	cautiously insert text previously deleted by 'ESC' 'RUBOUT'.
ESC p	delete marked text and write it into the scratch file. Can be reproduced elsewhere with 'ESC' 'g'.
ESC d	duplicate: copy marked text into scratch file (PUT " "), subsequently turn off marker. Can be duplicated as often as desired.
ESC g	write the text deleted with 'ESC' 'p' or duplicated with 'ESC' 'd' at current cursor position, i.e. insert scratch file at current position (GET " ").

Writing characters[1)]

ESC a	writes å.
ESC A	writes Å.
ESC o	writes ö.
ESC O	writes Ö.
ESC u	writes ü.
ESC U	writes Ü.
ESC s	writes ß.
ESC (	writes a [.
ESC)	writes a].
ESC <	writes a {.
ESC >	writes a }.
ESC #	writes a # that can also be printed.
ESC -	writes a (protected) separator, see text cosmetics.
ESC k	writes a (protected) "k", see text cosmetics.
ESC blank	writes a (protected) blank, see text cosmetics.

Assign command to key

ESC ESC	turn on command dialogue
ESC ! taste	in the command dialogue: assign written command to key
ESC ? taste	in the command dialogue: display command assigned to 'key' for editing.
ESC k	in the command dialogue: display the command last edited (single – line ELAN program).

A detailed description of the command dialogue can be found in the following chapter.

1) This is the standard pre – assignment of these keys but they can be changed by users and in user programs.

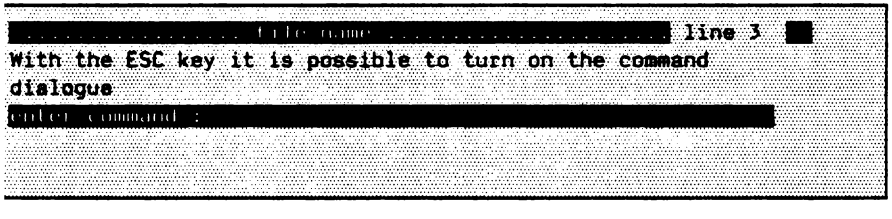
4.3. The most important Editor commands

4.3.1. The command dialogue

Some operations can only be performed with difficulty using the keys we have described so far. It is, for example, very time – consuming to find a certain position in a text. Other operations cannot be performed at all using the keys described in the previous chapter. Examples of such operations include setting the line width or calling programs to process the file to be edited. Such operations can be carried out by commands you enter while being in the Editor.

In order to enter commands in the Editor we go into the command mode.

Pressing 'ESC' twice the following request appears in the Editor:



A command line in which the user can write commands appears on the screen. Pressing the 'CR' key executes the command.

4.3.2. Selecting line and text position

On the command level of the Editor you can enter commands to position anywhere in the file.

You have edited a (larger) text and are facing the problem of finding the corresponding text positions for correction.

Example:

Looking through a print copy of your text you notice that you have made a spelling mistake. Instead of "these characters" you have written "these chatacters". In order to select the text position you proceed as follows: you position to the beginning of the file and press the key sequence

ESC ESC

On the screen appears:

Now you write the text passage that is to be found:

By specifying a TEXT in double quotes, the system searches for the enclosed TEXT 'these chatacters' beginning from the current cursor position. If 'these chatacters' is found, the cursor stops at the text being searched for. Otherwise the cursor stops under the last line of the file (end of file).

Another possibility exists to get to a remote text position:

On the screen appears:

Now you state the line number you are searching for:

This command positions the cursor on the 134th line.

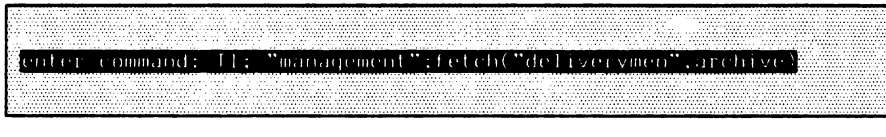
4.3.3. Searching and deleting

Both on the command level of the Editor and the Monitor you can enter any commands. These can be linked to (ELAN) programs. For generating these programs you edit in the command line as usual. The system provides you with commands for positioning, searching and replacing within your ELAN program. Any ELAN programs are allowed.

The command line can be edited like a "normal" text line (positioning, overwriting, inserting, deleting and marking). Before a program generates an output or before erroneous commands cause error messages, the cursor is positioned to the left upper edge. In order to retain to messages, you should enter the 'pause' command. These messages are then displayed in the first line of the screen. You then return to the Editor and can work as usual.

Commands are separated from each other by a semicolon.

Example:



Your ELAN program consists of three commands: first you position on the first line and the system then searches for the word "management" (from the first line onwards). Then the "deliverymen" file is read from the archive into the main memory.

The two commands described (selecting a text or a line) are special commands and cannot be combined in this form with other commands (with a semicolon). For this reason there is an ELAN form which allows them to be used together with other commands:

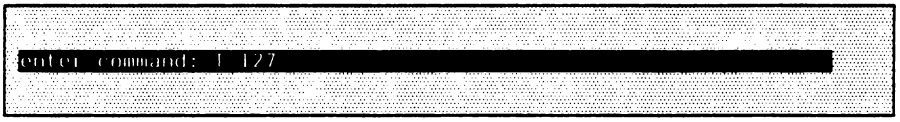
a) Search for TEXT from the current cursor position onwards ('D' is an abbreviation for 'DOWN'):


```
(* short version *)
```


```
(* general version *)
```

b) positioning on a line ('T' is an abbreviation for 'TO LINE'):


```
(* short version*)
```




```
(* general version *)
```

Several commands can be specified in the command line. In this instance, the individual commands have to be separated from each other by ';'.

Example:

turns on the command mode.

These two commands are executed in sequence. First the cursor is positioned on the first line and the system then searches for 'another character' (from the first line onwards). Thus, we can scan the file not only from the current line but also the entire file. If we do not want to search towards the end of the file, but rather towards the beginning of the file (i.e. 'upwards') we can use the 'U' command (abbreviation for 'UP').

Another useful command is the 'C' command (abbreviation for 'CHANGE'), which searches for a TEXT and then replaces it.

Example:

ESC ESC

The system searches for 'old characters' from the current cursor position onwards. If the TEXT is found, it will be replaced by 'new characters'. However, if 'old characters' is not found in the file, the cursor is at the end of the file (like in the case of unsuccessful searching with 'D').

Like all the other commands, the 'C' command can be combined with other commands.

Example:

ESC ESC

Here, the system searches for 'misspIling' after the 500th line and replaces it if appropriate. If a TEXT is to be replaced not only once, but each time it occurs, use the 'CA' command (abbreviation for 'CHANGE ALL'):


```
enter command: "this old text" CA "this new text"
```

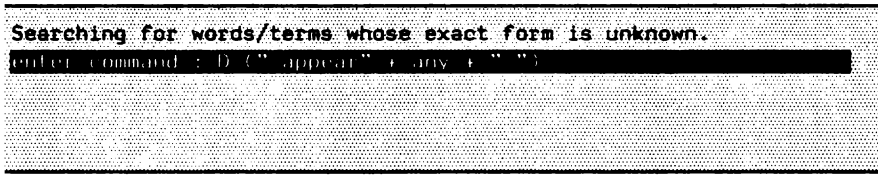
This replaces 'this old text' by 'this new text' every time it occurs after the current cursor position.

4.3.3.1. Pattern Matcher

The pattern matcher is a tool for pattern recognition. It serves the description of texts which can appear in different forms. When searching for or deleting a text, it is not given in a fixed form, but a description of its desired structure is given.

It may often be the case that you search for texts or want to replace texts that can appear in several variants within a longer text.

Example: We are searching for 'appear' in different combinations, i.e. 'appears' or 'appeared' as well. All text positions that correspond to this pattern can be found in one search method by using the pattern that describes these texts for the search:



How to read this:

Search for 'appear', followed by any characters plus one blank or just one blank.

This search command is successful with 'appear', 'appears, 'appeared' etc.

4.3.3.2. How to construct a pattern?

Texts are described by their construction pattern of known and unknown parts.

A text which is not known in its definite form but whose construction can be described by a pattern consists of parts that are called :

- known texts
- unknown texts

and that can be combined with the operators:

'+'	assemble
'OR'	alternative

A known text, for example, is a part of a wanted text that can be regarded as definitively given like, for instance, the word stem 'appear' in the above example. As usual, such a known text, which is placed in double quotes, is noted as TEXT Denoter "text".

On the other hand an unknown text is of a form that is not to be described in greater detail. The pattern that describes an unknown text stands in for any variety of texts which correspond to this pattern.

The procedure:

```
any
```

supplies the pattern of any text.

In the introductory example the word stem is known, the partword 'appear' can be stated 'in clear language'. The endings differ, depending on the context in which the word we require appears , i.e. unknown for the time being.

Such an unknown text can either be described by listing the possible alternatives of its appearance or by the 'any' procedure.

```
("text" + ("s" OR "ed" OR "ing" OR ..... )
```

alternative combination by OR

```
("text" + any + ....            )
```

additive combination by +

One has always to be aware that the search process of the pattern matcher examines character strings and not just words and that the longest possible word is searched for.

A badly described search text not only costs a lot of calculating time but also causes undesired results: for example the article 'the' should be searched for with a leading blank as ' the', otherwise each word that contains the syllable 'the' produces a hit in the search process.

Since the search for unknown texts could produce many undesired results, the 'any' procedure can be modified in two ways:

```
D(" t" + any (2) )
```

The length of the unknown text is default by stating the number of characters of which the text consists. This entry stands in brackets behind 'any'. (In this example exactly two characters).

```
D(" t" + any ("eirsmh"))
```

The alphabet, of which the unknown text can consist, is stated. (In this example the text that produces a hit can only consist of the characters 'e', 'i', 'r', 's', 'm', 'h', for example the, their, them, etc.

```
D(" d" + any (4,"eirsmh")
```

A combination of the limitations is possible. (Now only 'this', 'the', 'them' produce a hit.)

NOTE: The character '*' takes a special position since it can be used as an abbreviation for 'any'. If this character is to be searched for or to be replaced, you have to write 'any (1,"*")' instead of "*".

Further information about the pattern matcher can be found in the EUMEL User Manual for programming.

4.3.4. Assigning commands to keys

Frequently used commands can be assigned to keys. This makes it possible to adapt the Editor to the special needs of a user.

Frequently used commands can be assigned to a key with the triple key sequence

ESC ! 'key'

Example:

ESC ESC (* the command line appears *)

ESC ! s (* the command 'save (SOME myself)' is now assigned to the 's' key *)

If the 's' key is pressed, the 's' character appears on the screen. The 'save' command is executed with 'ESC' 's'. More complex commands can of course be assigned to keys.

If you want to change a command which has been assigned to a key, you press the triple key sequence

ESC ? 'key'

in the command dialogue.

Example:

(* enter the command dialogue *)

(* 'save (SOME myself)' appears *)

This command can now be altered and executed (with 'CR') or again be assigned to the same or another key (with ESC ! 'key').

In the Editor the last entered command in the command dialogue can be repeated by means of 'ESC' 'f'.

4.3.5. Using texts from other files

Sometimes it is necessary to write a text into another file (e.g. if you want to use this text again) or to insert a text from another file into the text to be edited. The 'GET' and 'PUT' commands make it possible to exchange texts between files (see also parallel editing).

The 'GET' – command enables us to copy texts from another file to the current write position.

fetches the text 'sender'. If you then write a lot of letters, you only need to write the sender once in the file 'sender' which you can then insert at various positions in the file using the 'GET' command (which you can assign to a key).

The 'PUT' command writes previously marked text passages into a file.

writes a marked text into the 'addresses' file. 'addresses' is created, if necessary. If the 'addresses' file already exists, you are asked whether the file can be erased in order to accept the marked text (overwriting). Otherwise the marked text is added to one already in existence in 'addresses'. Repeated marking and the 'PUT' command can therefore be used to collect texts from one file and to put them into a new file.

4.3.6. Processing longer lines

The Editor is set for a line width of 77 characters. It is often necessary to write using another line width which can be set using the 'limit' command. This, however, also makes positioning the cursor within a line somewhat different because lines that are longer do not fit on the screen all at once. In such a case the lines are scrolled.

Another line width can be set using 'limit'. Note that the set line width applies to the whole file.

Example:

Now you can write as usual. The current line, however, is not overrun at the end of the screen, but only when column 180 is reached (unless you have already terminated it with the CR key). If you write past the right screen edge, the cursor remains at the end of the screen but the line is shifted to the left, i.e. "scrolls" to the left (the beginning of the line seems to disappear to the left).

Positioning the cursor is similar. If the cursor is positioned beyond the right screen edge, the line is likewise scrolled. 'HOP' 'RIGHT' causes paging to the right within a single line. The situation is analogous for a shifted line when the cursor is positioned to the left ('LEFT' or 'HOP' 'LEFT').

When writing tables, it is sometimes useful to set the window at another starting position (other than 1). This can be done with the 'margin' command.

Example:

The Editor window now displays a section of the file starting with column 50. "M50" is displayed in the header line.

4.3.7. The most important commands

Some commands are especially programmed for word processing in the Editor. The most important ones are introduced in this section.

any

TEXT PROC any
Gives a pattern of any form and length (i.e. also for the length 0) for search operations.

```
"pro" + any + "mer"
```

any

TEXT PROC any (TEXT CONST alphabet)
Gives the longest possible text which consists of the characters stated in 'alphabet'.

```
any ("1234567890")     (* search for numbers *)
```

any

TEXT PROC any (INT CONST length)
Gives a pattern of any form and the length 'length'.

```
" t" + any (2)
```

any

TEXT PROC any (INT CONST length, TEXT CONST alphabet)
Gives a pattern of the length 'length', which only consists of the characters of 'alphabet'.

```
" t" + any (4,"mheris")
```

C

OP C (TEXT CONST pattern, replacement)
'pattern' is searched from the current position onwards in the direction to the end of the file and it is replaced by 'replacement'. The cursor then stands behind 'replacement'.

```
"old" C "new"
```

CA

OP CA (TEXT CONST pattern, replacement)
Works analogously to 'C' from the current position. The action, however, is repeated until the end of the file is reached. After the execution each 'pattern' is replaced by 'replacement'. The cursor then stands at the end of the file.

```
"old" CA "new"
```

D

OP D (INT CONST n)
Positions the window n lines forward towards the end of the file.

```
D 50
```

OP D (TEXT CONST pattern)
Searches for 'pattern' forward in the direction to the end of the file. The search starts directly behind the current cursor position. If 'pattern' is not found, the cursor stands at the end of the file. If 'pattern' is found, the cursor stands directly on the first character of 'pattern'.

```
D "pattern"
```

GET

OP GET (TEXT CONST file name)
Copies the contents of the file with the given name in front of the current cursor position. If a section of the source file is marked, only this section will be duplicated.

```
GET "source file"
```

OP G (TEXT CONST file name)
Same as GET.

limit

OP limit (INT CONST limit)
Sets the right margin on 'limit'.

```
limit (50)
```

margin

PROC margin (INT CONST start)
All lines appear from the column 'margin' in the window.

```
margin (50)
```

OR

TEXT OP OR (TEXT CONST textone, texttwo)
Gives a pattern if 'textone' or 'texttwo' is found. The sequential order is of no importance.

```
D ("business" + ("man" OR "woman"))
```

PUT

OP PUT (TEXT CONST file name)
Sets up a file with the specified name and writes the marked text passage into it.

```
PUT ("my scratch file")
```

OP P (TEXT CONST file name)
Same as PUT.

T

OP T (INT CONST n)
Positions on line 'n'.

```
T 999
```

type

PROC type (TEXT CONST character string)
Inserts 'character string' at the current position in the file being edited. Particularly useful in combination with the 'code' procedure in order to get the characters in the text that are not engraved on the keyboard.

```
type(code(200))
```

U

OP U (INT CONST n)
Positions the window n lines towards the beginning of the file (backwards).

```
U 100
```

OP U (TEXT CONST pattern)
Searches for 'pattern' towards the beginning of the file (backwards). The search starts to the left of the current cursor position. (cf. 'D')

```
U "pattern"
```

word wrap

PROC word wrap (BOOL CONST on)
Turns the automatic word wrapping function on (pre – set) or off.

```
word wrap (true)          (* turned on *)
word wrap (false)         (* turned off *)
```

4.4. Possible errors and how to remedy them

As a beginner you may sometimes get into situations in which you will not know what to do. In the following examples you will find a number of tips to help you out in difficult situations.

What can I do if ...

after

```
continue ("task name")
```

the monitor does not say

```
enter command:
```

but "remains silent"?

=> When you used the task last you did not leave it with the 'break' command (maybe you pressed 'SV'?). You are in the Editor now but do not see the text passage edited last. Press the

keys and the text will be newly displayed on the screen.

the Editor does not accept pressed keys?

=> You have pressed the 'STOP' key (often implemented as 'CTRL' 'a', depending on the unit you are using) by mistake, i.e. stopping the screen display.

Press the 'PROCEED' key (= 'CTRL' 'c', i.e. continue the screen display). All keystrokes which have not brought up any results in the meantime are now displayed.

'STOP' and 'PROCEED' can be assigned to other keys, depending on the keyboard.

the learning mode has been turned on (by mistake) for a long period of time?

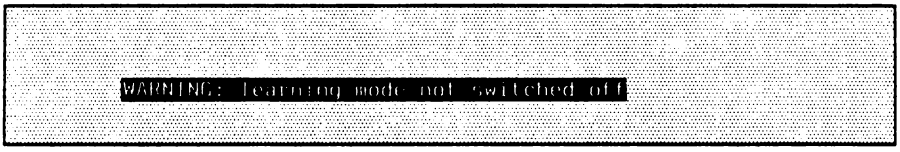
=> a) You suddenly realize that over an indefinitively long period of time all your keystrokes have been learned (to be seen from the 'LEARN' display in the header line).

What can you do?

With the

command you forget everything that has been learned and turn off the learning mode.

=> b) You leave the Editor with 'ESC' 'q' and the message



appears on the screen.

What can you do?

You can forget what has been learned at once with

you have set too many paragraph markers in your text and have to delete them?

=> You position to the line in which the paragraph marker is to be deleted. Then you press the 'TAB' key in order to position the cursor behind the text, then the 'RUBOUT' key. When you leave the line upwards or downwards, the paragraph marker disappears.

after

save ("file name","father task")

the operating system does not react any more?

You have not prepared the father task with the 'global manager' command in that process for receiving data from other processes.

you want to call the archive in your task with the

archive ("archive name")

command and the system gives you the following message

"error: archive is used by "bib" task"

=> There are two possibilities:

a) Another user requires the archive drive at this moment. You have to wait until he has finished his work.

b) Another user (or you yourself) forgot to release the archive with the

release (archive)

command in that task. If it was you, you make up the command. Otherwise the 'archive' command can be entered successfully if the archive has not been accessed for five minutes.

you have learned a (seemingly or real) endless loop on a key (e.g. the "x" key) and activate it (by mistake or consciously) with 'ESC' 'x'?

=> As always, when you want to terminate an endlessly running task, you get into the supervisor mode with the 'SV' key and you terminate the endless loop with the

'halt'

command.

With

the learned elements are 'forgotten'.

you want to leave your file and

(seemingly) does not function?

=> You have pressed 'SHIFT LOCK' / 'CAPS LOCK' by mistake and pressing 'ESC' 'q' shows no effect (as other key combinations with capital letters might not show any effect either).

PART 5: Text cosmetics and printing

5.0. Preface

The text cosmetics programs of the EUMEL system provide easy to learn and to use means of preparing texts for printout (technical term: formatting) and techniques for manipulating them.

The text cosmetics programs process your files created by the EUMEL Editor. For this reason, you should familiarize with the EUMEL Editor first of all.

The program structure is such that most tasks are controlled by directives inserted into the text. Such instructions for text cosmetics and the EUMEL printer will be called in short *'directive'* in the following. The form of directives is the same both for text cosmetics and the EUMEL printer and corresponds to the ELAN syntax. Be careful to note the difference between a command and a text directive: while a command is executed immediately, a directive imbedded within the text only becomes effective after calling the text cosmetics and printer programs.

The functioning of text cosmetics directives is easy to learn and, moreover, can be learned step by step. Therefore, a helpful piece of advice for beginners: Please glance over this part of the user manual first so that you get a rough idea of the facilities offered by the text cosmetics programs. Then you can select those parts of the text cosmetics which are required for your special application and use them when needed.

5.1. Introduction to using text cosmetics

This chapter provides an overview of the available text cosmetics programs.

Writing, formatting and printing texts

In the EUMEL system we differentiate between three steps of editing a text: **preparation**, **formatting** and **printing**. The advantage of this division into different work stages being that you only have to concentrate on one step at a time.

Text preparation or word processing

Texts are written using the Editor. In this step of text preparation you can concentrate solely on writing your text and checking its contents for correctness. If a text is printed without directives, then it appears as if it were written using the Editor. Text cosmetics directives can also be inserted into the text when it is being prepared.

```
It is essential that you read the chapter 'Editor'
#on("b")#very#off("b")# thoroughly.
```

Printout:

It is essential that you read the chapter 'Editor'
very thoroughly.

Texts should be prepared in 'continuous text' mode, i.e. words which would run over the end of line are carried over into the next line by the Editor without hyphenation.

Text cosmetics or text formatting

After you have written a text it can be formatted by text cosmetics programs without changing its contents. This may also be done before or after any corrections. Text cosmetics currently offers four programs which can be used as required:

- **'lineform'/'autoform'** formats a text line by line and carries out hyphenation. 'lineform'/ 'autoform' furthermore permits the use of different type fonts and font heights.

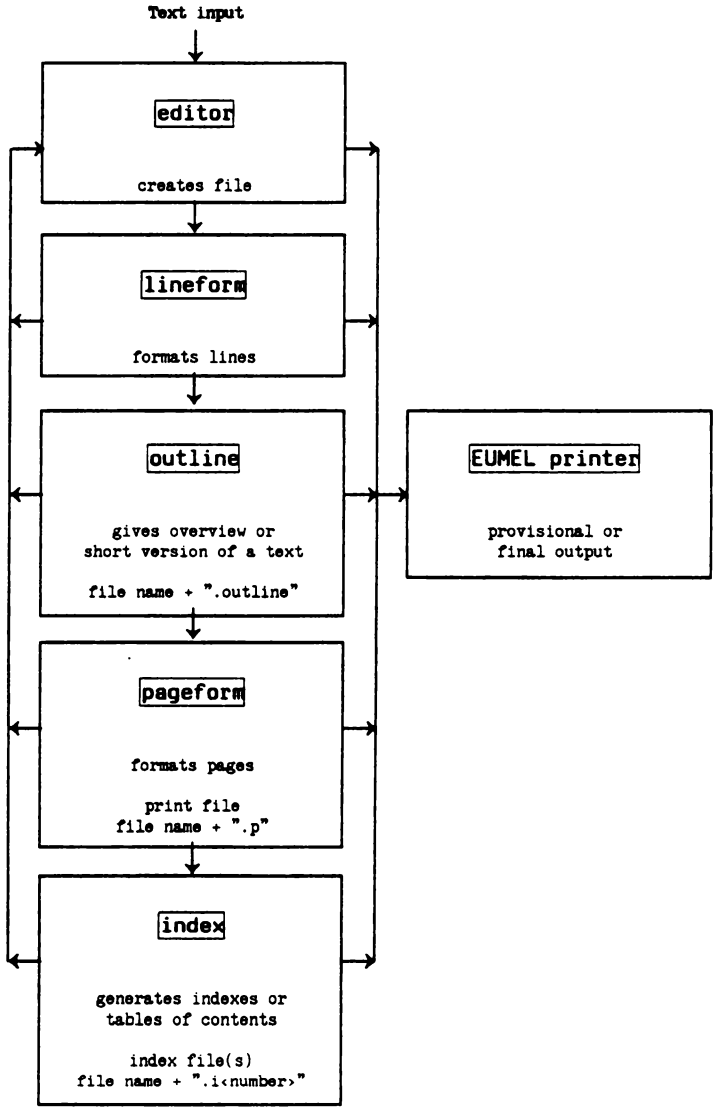
- **'pageform'/'autopageform'** permits the formatting of texts into pages (technical term: "page makeup"). Here 'pageform'/'autopageform' takes different font sizes into consideration. With 'pageform'/'autopageform' you can determine the division of a text into pages, format a page into columns ("newspaper form"), insert lines at the beginning or end of each page, preserve page numbering ("technical term: pagination") and format footnotes.

- - - **'index'** permits the creation of indexes and tables of contents of a file that has been processed by 'pageform'/'autopageform'.

- - - **'outline'** fetches all headings and keywords marked/flagged with an 'index' directive out of a file. Thus, it creates an overview or abridged version of a text.

Printing

Texts can be printed at any time during text processing. The EUMEL printer observes the same directives as the text cosmetics programs and some additional ones which are only necessary for print editing. Special print features, such as different type fonts, can only be generated on special printers. If a printer lacks specific hardware characteristics, the function requested by the user is ignored. Therefore it is possible to make provisional printouts for corrections on low-priced printers, too. (cf. 5.6.1.)


Text input
editor
creates file
lineform
formats lines
outline
gives overview or
short version of a text
file name + ".outline"
EUMEL printer
provisional or
final output
pageform
formats pages
print file
file name + ".p"
index
generates indexes or
tables of contents
index file(s)
file name + ".i<number>"

5.1.1. Directives for text cosmetics and printer

This section describes how you can insert directives for text cosmetics and printer programs into a text. Note that each directive has to be enclosed in '#' characters. If the '#' character is required in your text, it must be written using 'ESC'.

There are two different kinds of directives:

a) Directives which change the entire layout of a manuscript (*"layout directives"*). Among these directives are the #limit (...)# (setting the linewidth), #linefeed (...)# (setting the line spacing), #page# (new page) directives etc. These directives are effective from the succeeding line onwards and therefore should be placed in an extra line between the text.

```
#type ("trium8")##limit (11.0)#
#start(5.0,1.5)#
#pagelength(17.4)##pagenr("%",148)##setcount(1)#
#block##pageblock#
#count per page#
#headeven#
#lpos(0.0)##cpos(5.5)##rpos(11.0)#
#table#
                    EUMEL User Manual
#fillchar(" ")#
#on("u")#       #off("u")#
#table end##clear pos#

#end#
#headodd#
#lpos(0.0)##cpos(5.5)##rpos(11.0)##fillchar(" ")#
#table#
     Part 5: Text cosmetics and printing
#fillchar(" ")#
#on("u")#       #off("u")#
#table end##clear pos#

#end#
```

The printout (the result of the directives) can be seen in the present User Manual.

Directives which are valid for the entire text must be placed at the beginning of a file.

b) Directives which are to effect the following text immediately such as #type# (type font), #on#/#off# (modifications such as underlining or bold face), #ib#/#ie# (marking index words) etc. Such directives become effective immediately and can be placed anywhere on the line (as in the following example).

```
#on("u")#Exceptions#off("u")# are especially mentioned
when describing the directives.
```

Printout:

Exceptions are especially mentioned
when describing the directives.

Further examples of text cosmetics directives:

```
#page#
#free(3.0)#
#type("quadrato")#
```

These directives correspond – like all commands in the EUMEL system – to the ELAN syntax (among other things they have to be written in lower case letters; parameters must be in paranthesis; several parameters are separated by commas; TEXT parameters in double quotes; REAL parameters with decimal point etc.). Blanks are of no importance (except in TEXT parameters) and can be used for better readability as desired.

The characters of a text directive are not included in a count when formatting a line or a page and they are not printed by the EUMEL printer. A line that only consists of directives is treated analogously even if it is ended by CR.

5.1.2. Calling the text cosmetics programs

This section describes how to activate the text cosmetics programs.

The text cosmetic programs are activated by commands (i.e. on the 'enter command' level).

```
enter command:
lineform ("file name")
```

```
or:
autoform      ("file name")
pageform      ("file name")
autopageform  ("file name")
outline       ("file name")
index         ("file name")
```

'lineform'/'autoform' can also be activated from the EUMEL Editor. For this purpose you mark the paragraph of the file that is to be formatted and in the command mode (press ESC ESC) enter 'lineform' or 'autoform' (without parameter).

The 'pageform'/'autopageform' program creates a print file from the input file which is assigned the name of the input file with the addition of '.p'.

```
enter command :
pageform ("file name")
```

As a result you get: "file name.p"

The 'index' program can only process a print file:

```
enter command :
index ("file name.p")
```

and generates the requested indexes in files which are marked with the addition '.i<number>'.

Examples: "file name.i1", "file name.i2" etc.

'outline' also creates a new file.

```
enter command :
outline ("file name")
```

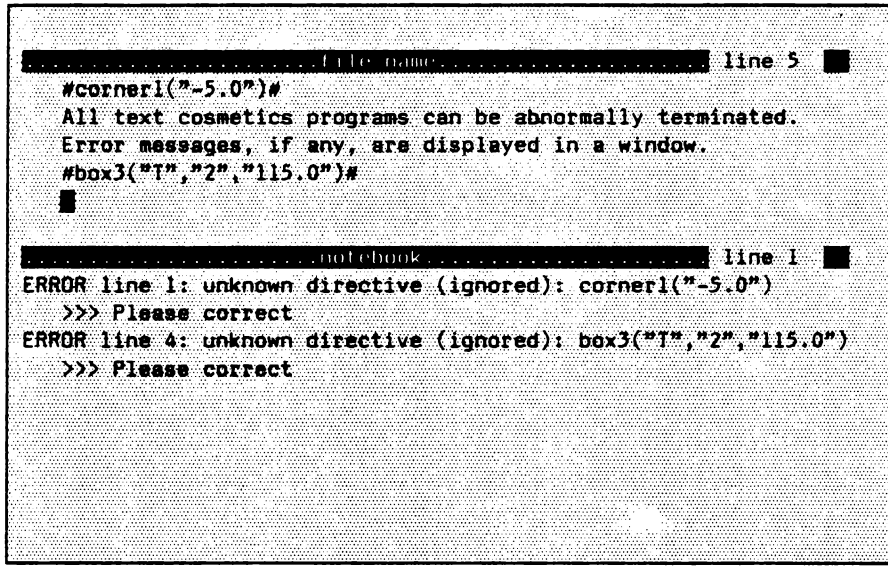
This leads to the following result: "file name.outline"

5.1.3. Abnormal termination and error messages

All text cosmetics programs can be abnormally terminated. Error messages, if any, are displayed in a window.

The text cosmetics programs can be abnormally terminated at any time by using the ESC key or the SV key and the supervisor command 'halt'. The input file remains unchanged at your disposal then. An abnormal termination may become necessary when a program with a wrong file has been called or if too many error messages were reported.

All text cosmetics programs report errors if directives are used incorrectly. The error messages are displayed on the screen. When a program has finished, the window Editor is automatically called if any errors have been discovered and the error messages are displayed in the lower window (the notebook) while the input file is displayed in the upper window for correction.



In order to change from the input file to the notebook – and vice versa – actuate

ESC w.

5.2. Lineform/Autoform

The 'lineform' or 'autoform' programs format a text line by line (with hyphenation, if required) taking into account type font and line width.

Two programs (commands) are supplied for line formatting which differ only in their interactive mode (treatment of hyphenation):

- **autoform:**
  Line formatting with automatic hyphenation. 'autoform' should only be used for texts in which a few hyphenation errors are of minor importance, e.g. for provisional printouts.

- **lineform:**
  Line formatting with hyphenation "by hand", the program suggests suitable positions for hyphenation (based on German division rules). The break point can be moved interactively depending on the space that is left on a line for the word to be split.

'lineform'/'autoform' have four main tasks:

- Filling of lines:
  'lineform'/'autoform' can be used particularly well after corrections where – after inserting or deleting – lines that are incomplete or too long may remain in the file.

- Creation of lines with different type fonts:
  If several type fonts (#type# directive) are used within one file, 'lineform'/-'autoform' calculate the number of characters which fit onto a line on the basis of the set line width.

- - - - Processing different line widths:
Sometimes it is necessary to change the line width (#limit# directive). This is taken into account by 'autoform'/'lineform'.

- - - - Hyphenation:
Automatic ('autoform') and interactive hyphenation ('lineform').

'lineform'/'autoform' accept a file as input and change this file. This requires an (internal) temporary file. Therefore, you should ensure that there is enough storage space on the system which, however, is only required temporarily for the formatting procedure.

Both 'lineform' and 'pageform' are pre-set on the first type font of the font table, i.e. a line width of 16.0 and a pagelength of 25.0. If the first directives, which could change this, are faulty, the original values are kept (the same applies to the ignored directives).

After the command has been entered 'lineform'/'autoform' inquire with which type font and with which line width the file is to be formatted. Doing so the pre-set directives appear first. Example:

```
LINEFORM (for ... lines): file name

font please   :  micro
linewidth (cm):  16.0
```

These directives can now be replaced by the ones needed by you. This information is stored in the file in the form of #limit# and #type# directives by 'autoform'/'lineform' and thus does not need to be respecified when the file is processed again.

If lines are longer than the specified line width, words that run beyond the line width will be carried over to the next line. Shorter lines will be filled up with words of the following line until the line width is filled. However, words are not moved beyond paragraph ends. Therefore, you should make sure that paragraphs are marked correctly before using 'lineform'/'autoform'. Missing marks should be inserted (CR at the end of a line), otherwise lines will be contracted beyond paragraph ends. This is particularly unpleasant with column lines.

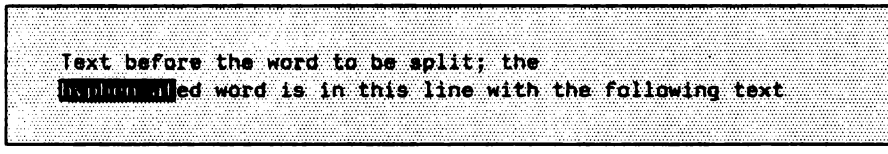
Indents (blanks at the beginning of a line) are also retained by 'lineform'/'autoform' when formatting lines.

5.2.1. Formatting line by line

5.2.1.1. Interactive hyphenation

'lineform' splits words interactively, i.e. 'lineform' suggests suitable positions for hyphenation which you can accept or reject.

If a word does not fit into a line entirely, it is offered interactively for hyphenation. In case of hyphenation, directives within the word are taken into account accordingly. The context of the word is also shown to facilitate the hyphenation process. The break point appears within the word at a position where it could be split.



The part of the word to be split that would fit onto the line is displayed inversely. You can move the break point with the aid of the positioning keys within the marked area. At the desired splitting position (the part of the word that is to remain on the line is to the left of the break point) the CR key can be pressed. CR instructs the 'lineform' program that the word is to be split at this position. 'lineform' adds the " – " character to the first part of the word and writes the divided part of the word into the following line.

The following operations are available in interactive hyphenation:

key	function
CR	split
←	move break point by one character to the left.
→	move break point by one character to the right.
HOP ←	set break point before the word (the word is not split at this position).
HOP →	set break point at the end of the marked area.
BLANK	split character is changed from "–" to " ". This can be used to split words which are not to be written together into two words during the hyphenation process.
–	changes the split character from a blank (" ") back to the splitting character ("–").
ESC	termination of 'lineform'/'autoform'. The file to be processed remains unchanged.

Two special conditions are to be noted in interactive hyphenation:

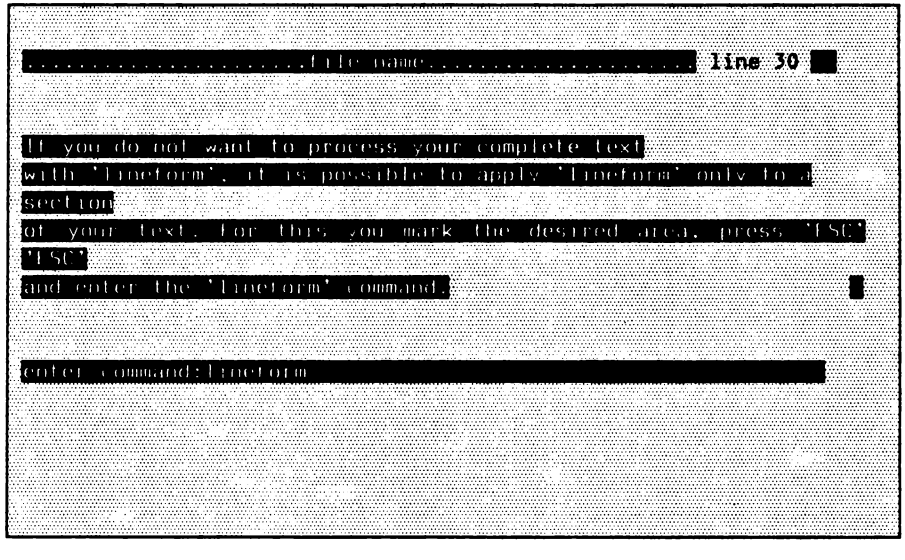
- In case of words with a hyphen the breakpoint after the hyphen is displayed as a blank.

- If a word is split between the characters "ck", the character "c" is changed into a "k". (German hyphenation rules)

  Example: Druk – ker

If necessary for line formatting, the 'lineform' procedure cancels existing hyphenation from a text (the split character is removed and the word parts are rejoined) if the split is no longer at the end of the line (because of corrections or a change in line width).

If you do not want to process your complete text with 'lineform', it is possible to apply 'lineform' only to a section of your text. For this you mark the desired area, press ESC ESC and enter the 'lineform' command in the Editor. (see page 5 – 9)



5.2.1.2. Automatic hyphenation with 'autoform'

'autoform' works just as 'lineform', but hyphenations are carried out automatically.

If hyphenations are necessary when formatting, they are carried out automatically by 'autoform'. The hyphenations made are written in the notebook. After formatting the processed file and the notebook are displayed in order to check the hyphenation. Automatic hyphenation is highly reliable, however, this only applies to German texts. Nevertheless, errors might occur, especially with compound nouns. Then you have to correct them afterwards using the Editor. (cf. also 5.8.4.)

5.2.2. Different type fonts

Different type fonts are requested with the #type ("type font name")# directive.

Different type fonts (briefly called fonts) can be processed with 'lineform'. Every type font has got a specific height and each character has got a certain width. All types are printed on one basic line.

There are two different kinds of type fonts: with **equidistant spacing** all characters are of the same width (as with a typewriter). **Proportional spacing** can be found in printed books. Here, different characters have also got different widths. The characters ".", "i", "!", for instance, are thinner than the characters "w", "o", "m" etc.

With the

```
#type ("font name")#
```

directive you can switch to another type font (also possible several times within a line). The font is valid until a new type directive is given.

```
#type ("micro")#Now we are writing with a
type font called 'micro'. And now
#type ("modern15")#we change to
another type font. Now #type ("modern12")#
we would like to use a bigger
font. In order to switch back to our
original type font we request #type
("trium8")# 'trium8'.
```

Printout (without 'lineform'):

Now we are writing with a
type font called 'micro'. And now
we change to
another type font. Now
we would like to use a bigger
font. In order to switch back to our
original type font we request
'trium8'.

Which type fonts are available on your system depends, of course, on the printer you use. Using the 'list fonts' command you can find out which type fonts exist.

Type fonts can be printed in a modified way, i.e. differently (cf. the following section). All modifications are turned off when giving a new #type ("font name")# directive.

5.2.3. Modifying a type font

The #on ("...")# – and #off ("...")# directives can be used to modify a type font in its appearance. The type is not changed, but it is printed differently. At present, underlining, bold face, italic, and reverse are possible (depending on the printer being used).

The #on#/#off# directive acts as a switch which turns the desired type font modification on or off. The #on# directive turns the modification on, #off# turns it off.

```
The EUMEL-System enables you to write

#on ("italic")#italic#off ("italic")#
#on ("i")#           #off ("i")#

and

#on ("underline")#underline#off ("underline")#
#on ("u")#                  #off ("u")#

and

#on ("bold")#bold#off ("bold")#
#on ("b")#        #off ("b")#

and

#on ("reverse")#reverse (white on black)#off ("reverse")#
#on ("r")#                                 #off ("r")#
```

Printout:

The EUMEL system enables you to write

italic

and

underline

and

bold

and

Please note the following:

a) A #type# directive always turns off a preceding modification, i.e. a change of type fonts makes any #off ("b")#, #off ("u")#, #off ("i")# and #off ("r")# directives superfluous.

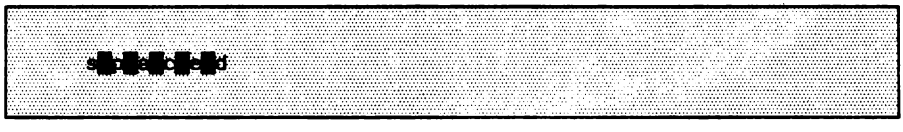
b) 'lineform'/'autoform' will generate a warning if you have forgotten to switch off a modification.

c) Not all types of printers are able to print the modifications stated here. It also depends on the printer which modifications can be used simultaneously.

5.2.4. Spaced writing

Hyphenation within a spaced written word is avoided by using a "protected" blank which is achieved with 'ESC' and 'blank'.

If you want to write a word s p a c e d, it is necessary to prevent this word from being split during formatting. Other words, such as in formulas, should also be written together on one line (e.g. 'sin (x)'). This can be achieved by not using a blank between the characters because a blank always indicates the end of a word to 'lineform'/'autoform'. Instead, you use ESC blank. For better identification the "protected" blank appears on the screen inversely displayed or as another character (depending on your terminal). However, a blank will appear in the printout.



Printout:

s p a c e d

5.2.5. Setting the line width

The line width can be set with the #limit(...)# directive.

The #limit# directive specifies the desired line width in cm. Note the difference between this directive and the Editor command 'limit', they have not got the same effect. The command specifies how many characters will fit into a screen line.

When first calling 'lineform'/'autoform' the user is interactively prompted for the line width and the type font which are then entered as #limit# and #type# directives in the first line of a file. The line width can be changed several times within a file.

The new line width is always valid from the line **following** the #limit# directive. Note that a number with a decimal point must be specified as a parameter in the #limit# directive.

```
#limit(9.0)#
        With the #limit# directive, it is
        easy to reformat paragraphs.
        The right writing margin is set by
        the #limit# directive while the left
        margin can be set by
        a corresponding identation.
#limit(11.0)#
```

Printout (formatted by 'lineform'):

> With the #limit# directive, it is easy to reformat paragraphs.
> The right writing margin is set by he #limit# directive while
> the left margin can be set by a corresponding identation.

The following list specifies useful 'limit' settings for the most common paper sizes:

format	'limit' (line width)	remaining margin
DIN A4	16.0 cm	2.50 cm each
DIN A5	12.0 cm	1.42 cm each
DIN A4 broadside	25.0 cm	2.35 cm each

5.2.6. Simple tables and lists

Lists and simple tables are automatically formatted and printed correctly if a few simple rules are observed.

If proportional spacing is used for writing tables, the columns are usually of different widths, even if the number of characters is identical in each column. This can be avoided by writing a "double blank" ("multiple blank"); there are table directives for more complicated tables (see also p. 5 – 31).

```
iiii  ooooo
mmmm  lllll
```

Printout:

iiii ooooo
mmmm lllll

First and second column are not aligned.

But, with double blanks:

```
iiii    ooooo
mmmm    lllll
```

Printout:

iiii ooooo
mmmm lllll

First and second column are now aligned.

The double blank instructs 'lineform'/'autoform' and the printer to calculate the individual positions and to take them into account when printing. This is only valid after a paragraph line. In some rare cases, especially when using type fonts that differ greatly in their sizes, the automatic table feature may not function so that columns are printed on top of each other. In such cases, the number of double blanks must be increased.

Practical tip:
Note that, in order for the "automatic table feature" to work properly each table line has to be a paragraph line. It is advisable to check these lines before printing or to have 'lineform'/'autoform' process the file. If as a result of line – by – line formatting two lines are merged into one, due to a missing paragraph mark, they can be easily "broken apart" again using the Editor (press HOP RUBIN , CR and HOP RUBIN).

Lists are treated in a similar way.

```
1)  This is the first list item.
    This sentence is printed justified.
2)  This sentence, too.
```

Printout:

1) This is the first list item.
   This sentence is printed justified.
2) This sentence, too.

In a situation like this, the printed text is normally indented correctly. The automatic list feature works only after a paragraph mark. But here is an example of a typical error:

```
#type("normal")#
1) list item
2) list item
3) list item

#type("big")#M1) list item
```

The indentation width depends on the type font that is valid before the line and it is kept throughout the whole paragraph.

Printout:

1) list item
2) list item
3) list item

M1) list item

The blank between 'M1)' und 'list item' is not sufficient to prevent an overwriting. This error can be avoided by placing the #type# directive in a separate line. It should be done like this (desired type font in the preceding line):

```
#type("normal")#
1) list item
2) list item
3) list item

#type("big")#
M1) list item
```

Printout:

1) list item
2) list item
3) list item

M1) list item

The exact rules are somewhat complicated so they are not discussed here in detail (see p. 5-89, #block# directive). If the automatic table feature does not work, which is very rarely the case, you can use double blank(s) or the table directives (see p. 5-31ff).

5.2.6.1. Table directives

Using the table directives of the text cosmetics, tables can also be easily written with proportional spacing.

It is very easy to write a table with equidistant spacing since the screen layout corresponds largely to the printout later on. With equidistant spacing each character has got the same width – thus you can "see" at which table position a new column starts.

It is a bit more difficult to write tables with proportional spacing because each character has a different width. Therefore you cannot "see" exactly how wide a column will be when printed. "Simple" tables can be created with the double blank (see p. 5 – 27). The following table directives have to be used for more complicated tables.

In order to generate a table you proceed as follows:

- The column positions of the table are defined by the following directives. Instead of the dots the corresponding parameters have to be given in the directives.

```
#l pos (...)#          (* left justified *)
#r pos (...)#          (* right justified *)
#c pos (...)#          (* centering *)
#d pos (..., ...)#     (* centering round a character string *)
#b pos (..., ...)#     (* right justification *)
#fillchar (...)#       (* fill characters between columns *)
```

Centering round a character string is to be understood as follows: The column is written right justified up to the start of the specified character string and from there it is written left justified.

- Then write the table. It has to be embedded in the following directives.

```
#table#


#table end#
```

The columns of the table have to be separated from each other by at least two blanks. In a table all columns must exist. If a column has to stay empty, then a "protected" blank has to be used for this column.

- Since the column positions remain existent (even after the #table end# directive) the #clear pos# directive is to be given directly behind the end of the table.

- Then 'lineform'/'autoform' can be performed.

```
#r pos (2.2)##c pos (3.8)##l pos (5.8)##d pos (8.8, ".")#
#table#
first column  second column  third column   fourth.column
right         centering      left           deci.mal
justified                    justified          .
1234          1234           1234             12.34
12345         12345          12345           123.45
123456        123456         123456         1234.56
#table end# #clear pos#
```

Printout:

first column	second column	third column	fourth.column
right	centering	left	deci.mal
justified		justified	.
1234	1234	1234	12.34
12345	12345	12345	123.45
123456	123456	123456	1234.56

Such tables can be written in #head#, #bottom# or within a footnote. However, it is not possible to define a footnote within a table. Way out: Splitting the table round the footnote.

5.2.6.2. Setting the table positions

Using #pos# directives a specified position within a table can be set, at the same time specifying how the column is to be printed.

```
#1 pos (5.0)##r pos (10.0)##d pos (15.0, ".")#
```

The above directive sets the first column of the table 5 cm from the margin (left justified). The second column ends 10 cm from the margin and it is to be written right justified. The third one is printed at the position 15 centering round the decimal point[1].

Note that an "overlapping" of columns may occur (in our example the first column can write into the second one). 'lineform'/'autoform' report a corresponding error in case of column overlapping.

For each column position one element of a line is taken. When writing in the Editor the elements have to be separated from each other by at least two blanks. The first element is printed on the first column position, the second element on the second etc. The chosen type font is taken with possibly one modification for printing. The type font and the modification can be changed within the table[2].

Note that the table positions remain existent until they are explicitly deleted. (#clear pos# directive, see p. 5 – 38).

1) Column position < 0.0 and column position > 'set limit' are not allowed.
2) The space between the columns is not modified (i.e. not underlined etc.)

5.2.6.3. Right justified printing within a column

The directive #b pos (...)# causes texts to be printed with right justification.

```
#l pos (0.0)##b pos (2.2, 8.0)##l pos (9.0)#
#table#
lst column  The middle column is printed with right  3rd column
lst column  justification to the printing position   3rd column
lst column  '8.0'. In order to get a paragraph in    3rd column
lst column  this column a protected blank has to     3rd column
lst column  stand at the end of the column.          3rd column
#table end# #clear pos#
```

Printout:

1st column	The middle column is printed with right	3rd column
1st column	justification to the printing position	3rd column
1st column	'8.0'. In order to get a paragraph in	3rd column
1st column	this column a protected blank has to	3rd column
1st column	stand at the end of the column.	3rd column

5.2.6.4. Filling table columns (fill chars)

With the #fillchar# directive space between columns can be filled.

Take the case of a bill to be issued. The goods are to be printed left justified at the printing position '0.0' and the amount right justified at the position '9.0'. Between the goods and the amounts a corresponding number of dots ('.') are to be printed. The following printout:

30 User Manuals.....DM 450,–
10 System Handbooks.....DM 150,–

is achieved by

```
#1 pos (0.0)##r pos (9.0)##fillchar(".")#
#table#
30 User Manuals          DM 450,-
10 System Handbooks      DM 150,-

#table end##clear pos#
```

With the #fillchar# directive the filling character(s) is (are) set. Thus an according number of filling characters (instead of blanks) are printed from the end of the text of a column up to the beginning of the text of the next column. The filling characters remain switched on until the #fillchar# directive is given again. In particular the filling character – as well as the set column positions – remains existent after the table end. The #clear pos# directive deletes – in addition to the column positions – the set filling character, too (sets the character back to ' ').

Note that the filling characters are printed directly (i.e. without any blanks between the column text and the filling characters). If a space between the column text and the filling characters is desired, a protected blank has to be added to the column text or one has to be placed before the following column.

The #fillchar# directive is valid for spaces between all columns. If only *one* space between columns is to be filled, the #fillchar# directive has to be given accordingly.

```
#l pos (1.0)##r pos (5.0)##r pos (10.0)#
#table#
1#fillchar(".")#    3#fillchar(" ")#   4
2#fillchar(".")#   17#fillchar(" ")#   6
#table end##clearpos#
```

Printout:

```
1.....3                    4
2.....17                   6
```

In this case switched on modifications are also valid for the space between columns.

5.2.6.5. Deleting table positions

With the #clear pos# directive all set positions are deleted.

If completely new positions are to be set, the

```
#clear pos#
```

is used without any parameter. It deletes all set table positions. Note that #clear pos# also deletes the filling character for the #fillchar# directive (a ' ' is pre-set again). A single table position can be deleted for example by

```
#clear pos (10.0)#
```

5.2.7. Indices and exponents

With the #u#, #d# and #e# directives exponents and indices can be written.

The #u# directive ('u'stands for 'up') changes to exponent writing until the #e# directive ('e' stands for 'end') is met. When writing exponents or indices the next size down in type font is automatically switched to (if available).

```
a#u#i,k#e#
```

Printout:

$a^{i,k}$

The #d# directive ('d' stands for 'down') is designed for writing indices and works analogously to the #u# directive.

```
a#d#i,k#e#
```

Printout:

$a_{i,k}$

The automatic shift to the next in size type font only happens if there is a next in size type font in the font table. Otherwise the set type font is kept for the exponent.

After the #e# directive the type font that was valid before the corresponding #u# directive is called. The #u# and #e# directives thus form brackets. Within a directive any meaningful text cosmetics directive can be given. Note that the directives within a bracket are not to change the linefeed. If, for instance, a #type# directive is written in a bracket, the index/exponent is printed with this type font but the printer assumes that the linefeed is not to be exceeded. Therefore it is advisable to use only a smaller type font within an index/exponent. As mentioned before the former used type font is switched back to after the end of the bracket.

The indeces/exponents brackets can also be nested.

```
a#u#up by 1 #u#another one up#e#back by 1#e# basic line
```

Printout:

a^{up by 1 another one up back by 1} basic line

The following **restrictions** have to be noted:

1. An exponent (index) is positioned in such a way that normally no overwriting with the proceeding (succeeding) line occurs.

2. With multiple exponents or indices or when switching to another type font within an exponent (index) or if a smaller type font cannot be changed to, the exponent or the index may exceed the "normal" line. In such a case overwriting may occur and it can be compensated by the #linefeed# directive.

3. An exponent or index bracket is to appear entirely on one line.

4. Exponents and index printouts together, i.e. which are to stand on top of each other, are not possible with the #u#/#d# directives at the moment. However, the following can be done:

```
a#u#exponent#d#index of the exponent#e##e#
```

printout:

$a^{\text{exponent}}_{\text{index of the exponent}}$

5. Double blanks are of no importance within such a bracket, i.e. they have the effect of two "normal" blanks and not of an implicit positioning. Within such a bracket blanks are not widened if the #block# directive was given.

6. Indices or exponents should not be used together with the modifications #underline# and/or #reverse# since an underlining of indices and exponents within an underlined line could lead to the following result:

Printout:

Indices and exponents $a_{i,k}$ $a^{i,k}$ should not be underlined!

5.3. Pageform

5.3.1. Page by page formatting

'pageform'/'autopageform' format a file page by page and also perform routine work, such as the positioning of footnotes, page numbering etc.

The 'pageform' program is called by the command

```
enter command :
pageform ("file name")
```

'pageform' generates a print file from the input file (e.g.: "file name"); the name of the print file is formed by appending '.p' (e.g.: "file name.p").

The print file generated by 'pageform' consists of the input file plus newly inserted lines, if any. #head#, #bottom# or #foot# directives may cause lines to be inserted. This increases the number of lines in the file.

It is possible to include page numbers in header or bottom lines. 'pageform'/ 'autopageform' automatically increment these page numbers when they encounter a page break and insert them at a position marked by the user. Cross references are also possible.

After inserting header or bottom lines or footnotes, if any, 'pageform' calculates the number of lines that will fit onto a page of the specified pagelength, the line spacing and the respective type height of the type font assigned (#type# directive). Subsequently, 'pageform' displays the calculated page break on the screen. The page break can be moved interactively to the desired position or blank lines can be inserted/deleted in order to obtain pages of equal length. Furthermore, it is possible to divide pages into columns ("newspaper print") and format them interactively.

When there are different type fonts within a line, the line height is automatically calculated on the basis of the biggest font. You have to consider the fact that at the beginning of a line the type font of the preceding line is valid.

5.3.1.1. Automatic page formatting

'autopageform' works like 'pageform', however, page breaks are automatically placed.

At first 'autopageform' searches for the arithmetic page break. If there is a paragraph, the page break is placed there. If not, 'autopageform' searches for a paragraph in the preceding four lines. If none is found, the page break is placed at the arithmetic end of page.

If the #pageblock# directive is given, then at first the last four lines of a paragraph are searched in order to terminate the page there. If none is found, the pagelength is tried to be placed beyond the arithmetic page break (4 lines). In such a case 'autopageform' considers the 'pagelength' directive by compressing the line spacing.

5.3.1.2. Interactively moving page breaks

This section describes the interactive possibilities of formatting pages offered by 'pageform'.

The respective page end calculated by 'pageform' is shown on the screen together with the current page number. The page break appears approximately in the middle of the screen and is marked as a line generated by 'pageform' which can also be seen in the print file after page formatting has been completed. The EUMEL printer does not print this line.

```
Several footnotes within a page are collected by 'pageform'/-
'autopageform' in the order in which they occur and are placed at
the end of the page. The user has to make sure that the footnotes
are properly separated from each other (e.g. by blank lines).
#page##---------- end of page 215 ----------#
The footnote might not fit onto the current page and therefore
has to be placed onto the next page by 'pageform'/'autopageform'.
'pageform'/'autopageform' assume that the footnote mark is in the
line immediately preceding the footnote and writes this line on
the new page as well.
```

Above the mark you will see the last few lines of the page which has just been processed and below it the first lines of the next page. Using the positioning keys the mark and thus the page break can be shifted upwards. This prevents logically connected text passages from being torn apart as well as "widows" (last line of a paragraph is put on the new page).

In interactive page formatting, the mark cannot be moved downwards beyond the calculated page end or upwards beyond the previously processed page break.

If, however, the #pageblock# directive (see p. 5-91) has been given, you are permitted to shift the page break mark a few lines beyond the arithmetic page end. Press CR and the printer will compress the line spacing on this page, if possible. When this happens you ought to make sure that the page end after a paragraph is always placed *in front of* any blank lines. Otherwise the blank lines at the end of the page are also counted and a corresponding space is kept free.

Within a footnote the mark cannot be shifted. In such a case the user is interactively asked whether the footnote is to be continued on the next page. If the request is denied, 'pageform' positions in front of the footnote. From there the page break can be shifted as usual.

However, if the request for a footnote makeup is answered positively, 'pageform' positions the page break within the footnote. The rest of the footnote is moved to the next page with a note ('Continuation of last page')[1].

If page formatting leaves blank lines at the beginning of a page (e.g. by positioning the page break between two paragraphs), 'pageform' automatically removes them from the print file. If blank lines are required at the beginning of a page, the #free# directive in combination with the #page# directive should be used.

In addition, you can insert blank lines into a page of the print file and/or delete any lines from it (cf. b)).

[1] With foreign language texts this note should be altered accordingly in the '.p' file after 'pageform'.

The following operations are available in interactive page formatting:

a) Move page break:

'pageform' calculates the "arithmetic" page break and displays it on the screen as a mark. The mark can be moved interactively:

key	function
CR	position page break here
↑	move page break one line upwards
↓	move page break one line downwards (if previously moved up or if the #pageblock# directive has been given
HOP ↑	move page break one screen upwards
HOP ↓	move page break one screen downwards
ESC	terminate page formatting

b) Inserting blank lines and/or deleting lines

If the text is not correctly positioned on the page after 'pageform' has been run, blank lines can be inserted or lines can be deleted from the page (in the print file). This may be useful, for example, if by deleting a line, a paragraph would fit onto the page or if, by inserting lines, a paragraph ends on the last line of the page. Often it is also desirable to have pages of equal length. In this case, it is advisable to insert or delete blank lines before chapters and paragraphs.

In order to insert and/or delete lines, the mark has to be moved to the position, as described under a), where the modification is to be performed.

key	function
HOP RUBIN	insert blank lines. Blank lines may be inserted instead of the mark by pressing CR (several times, if necessary). HOP RUBIN ends the procedure (same as inserting lines in the Editor).
HOP RUBOUT	delete line. The line in which the mark is standing is deleted.

Afterwards 'pageform' calculates the page anew.

c) Confirm/delete #page# directive

If the 'pageform' procedure encounters a #page# directive, the desired page break is displayed on your screen. The #page# directive can either be confirmed or deleted.

key	function
CR	confirm page break
RUBOUT	ignore #page# directive. The 'pageform' procedure will then continue processing the file as if no #page# directive had been encountered.
ESC	terminate page formatting

5.3.2. Setting the page length

'pageform'/'autopageform' are preset to a page length of 25.0 cm (corresponding to a DIN A4 text field). If a different page length is required, the #pagelength# directive has to be inserted into the text.

```
#pagelength (20.0)#
```

sets the page length to 20 cm.

Note that

1. the newly set page length is only valid from the next page onwards (the hitherto set page length is still valid for the current page).

2. The set page length at the beginning of the file (i.e. before the first text line) is valid for the first page.

3. The decimal point has to be stated together with the page length.

The following table gives the most common German paper sizes:

format	page length (in cm)	upper and lower margin
DIN A4	25.0	2.35 cm each
DIN A5	18.0	2.15 cm each
DIN A4 broadside	16.0	2.50 cm each

5.3.3. Setting the line spacing

The #linefeed# directive sets a line spacing in accordance with the font height of the assigned type font.

'pageform'/'autopageform' always calculate the number of lines per page on the basis of the set type font. If a font has been selected which is twice as high as that of a typewriter, then as a result fewer lines will fit onto a page. Normally, you need not worry about this calculation procedure.

This is not true if a line spacing that differs from the "normal" spacing between lines is to be selected. In this case the #linefeed# directive is given. The parameter indicates by how much the line height is to be increased or decreased ***from the next printable line*** .

```
#linefeed (2.0)#
```

prints the following lines with double spacing. When this command is encountered, the line height is calculated as set font size multiplied by 2. The vertical line spacing is increased accordingly since the font size remains the same. This corresponds to double spacing on a typewriter (disregarding the fact that different font heights are considered here as well). 1 1/2 line spacing can be set with

```
#linefeed (1.5)#
```

```
#linefeed (0.5)#
```

sets the line height = set font size multiplied by 1/2 so that the lines will be printed partly overlapping (which will cause illegible results with some printers). If #linefeed (0.0)# is set, lines are printed on top of each other (printer – dependent).

Note that the specification in the #linefeed# directive is given in relative numbers. All other text cosmetics directives require specifications in cm. Thus the #linefeed# directive is an exception to the rule.

5.3.4. Keeping space free

With the #free# directive a continuous section of a page can be kept free.

The #free# directive is inserted into the text at positions where drawings, tables and the like are to be added after it has been printed. It can also be used between paragraphs, chapters etc. if the spacing does equal the multiple of the line height. The space specified in the #free# directive will be kept free.

```
#free (2.0)#
```

keeps two centimeters free. If the required space does not fit onto the page, it is reserved on the next page ('pageform'/'autopageform' set the page break before the #free# directive).

Practical tip:
The #free# directive should be written on a separate line so that it can be deleted by 'pageform' interactively if the #free# directive is awkwardly placed at the beginning or end of a page.

5.3.5. Starting a new page

In some places in the text, e.g. at the beginning of a new chapter, it is mandatory to start a new page. This can be done with the #page# directive.

In this case 'pageform' reports after how many centimeters on the page the directive was encountered. Then the page break can either be confirmed with CR or the directive can be deleted (in the print file). In the latter case, 'pageform' will recalculate the page as if the #page# directive had not been given.

At the same time, a new page number for the page may be set by the #page# directive (cf. the following sections).

The #page# directive has to be placed in a separate, otherwise empty line.

5.3.6. Headers and bottom lines

With the #head# and #bottom# directives, it is possible to insert lines at the beginning or end of every page.

Lines at the beginning ("headers") and at the end ("bottom lines") of every page are only written once and are marked by directives. 'pageform'/'autopageform' insert these lines at the corresponding positions.

```
#head#
                Our EUMEL User Manual

#end#
```

'pageform'/'autopageform' place this line (i.e. that enclosed in the #head# and #end# directives) at the beginning of every page in the print file.

The same applies to bottom lines which have to be enclosed in #bottom# and #end# directives:

```
#bottom#
                Author: I. Listig

#end#
```

Practical tip:
Insert at least one blank line at the end of a #head# or at the beginning of a #bottom# in order to separate the actual text from the headers or bottom lines.

'pageform'/'autopageform' count the pages, starting with page number '1'. (The following section describes how to include page numbers in headers or bottom lines). It is possible to create different headers and bottom lines for even – and odd – numbered pages (as in this User Manual). The #headeven# and #headodd# directives are used for pages with even and odd page numbers; #bottomeven# und #bottomodd# ditto. These directives always have to be ended with an #end# directive, too.

Headers and bottom lines may be changed several times within a file in order to provide different headings (e.g. for chapters). However, this should only be done at the beginning of a new page, i.e. directly *after* a #page# directive.

```
#page#
#head#
                New page head

#end#
```

Headers and bottom lines should be of identical appearance throughout the text, regardless of the directives given in the remaining text. For this reason, the current values for

```
limit
type
linefeed
```

assigned when defining a header or bottom line, are taken into account when inserting the lines. It is possible to use a different type font for header or bottom lines (than in the remaining text) by entering the #type# directive within the #head# or #bottom# area. Note that after leaving #head#, #bottom# and also #foot# areas the above mentioned directives are not automatically reset. Therefore, the directives used in the remaining text should be reset before the #end# directive.

```
#bottom#
#type ("small")#
                Author: I. Listig


#type ("normal")#
#end#
```

```
(reset
 type font):
```

5.3.7. Numbering pages

The '%' character representing the current page number is contained in the headers and bottom lines.

If the '%' character appears within a header or bottom area, 'pageform'/'autopageform' will insert the current page number on every page when inserting these lines (the page number is inserted each time a '%' character is encountered).

```
#head#
                    page: - % -
#end#
```

If the page number is to be placed in the center or at the right margin, the #center# (see p. 5 – 93) or #right# directives (see p. 5 – 94) can be used.

By creating a bottom area the page numbers can also be placed at the lower bottom of a page. Note that the line width increases when multi – digit page numbers are used.

In order to mark the existence of a follow – up page in a bottom area, the '%' character has to be written twice.

```
#bottom#

#right# %%
#end#
```

In the above example the page number is printed right justified.

It is sometimes necessary and advisable to keep a text in several files. In this case, page numbers must be reset in each follow-up file. This is performed using the #pagenr# or the #page# directive.

```
#page (4)#
```

starts a new page. The page number of the new page is '4'.

For some special applications, more than one page number may be required, e.g. if pages of the text as a whole as well as the pages in each chapter are to be counted separately.

```
#page (4711)#
#pagenr ("$", 1)#
#head#
          My book          page: %  chapter page: $

#end#
```

The #pagenr ("$",1)# directive is used to start a new page numbering from the following page onwards. '$' stands in for the new number. '1' means that the numbering starts with '1'. 'pageform'/'autopageform' will increment it by '1' with every new page and, if applicable, insert it into the headers and bottom lines. Two additional page symbols (other than the %) are possible.

Note that the new page numbers are only valid from the next page. If the #page (...)# or the #pagenr (...,...)# directive is given at the beginning of a file (i.e. before the first text line), the new page numbers are valid for the first page.

5.3.8. Writing footnotes

Footnotes are directly marked in the text by the #foot# and #end# directives. 'pageform'/'autopageform' put the footnotes at the end of a page.

Footnotes are written directly into the text by the user, preferably at the position where the footnote is to be called later. 'pageform'/'autopageform' place the footnote at the end of a page or before the bottom lines, if any. The user is requested to insert footnotes and mark them in the text. However, 'pageform'/'autopageform' will insert underscores before the footnotes when inserting a footnote at the end of a page in order to separate them from the running text.

```
#foot#
*) This is the first footnote on this page.
#end#
```

Printout:

---

*) This is the first footnote on this page.

Several footnotes within a page are collected by 'pageform'/'autopageform' in the order of their appearance and are placed at the end of a page. The user has to make sure that the footnotes are properly separated (e.g. by blank lines).

It is possible that the footnote does not fit onto the current page and then would have to be placed by 'pageform'/'autopageform' onto the next page. 'pageform'/'autopageform' assume that the marking of a footnote stands in the line immediately preceding the footnote and bring this line onto the new page as well.

```
It is also possible to write a footnote within a paragraph
as, for instance, in this line#u#*)#e#.#foot#
#u#*)#e# footnote within a paragraph
#end#
Afterwards you continue writing without
any interruption.
```

Printout (after lineform):

It is also possible to write a footnote within a paragraph as, for instance, in this line*). Afterwards you continue writing without any interruption.

In this case it is desirable that 'lineform' fills the line that precedes #foot# with the line that follows #end#. The following conditions have to be fulfilled:

1. Nothing can stand behind #foot#, i.e. no paragraph mark either.

2. Words from the line behind #end# are placed before the #foot# directive until the line is filled or the line behind #end# is emptied.

3. Note that text cosmetics directives also effect the footnote. If, for example, a #type# directive is concerned, the layout of the footnote can be changed. Therefore it is advisable to place any directives that are to change the footnote within the footnote.

*) footnote within a paragraph

Try to avoid writing extensive texts in footnotes (e.g. longer quotations). Because of program technical reasons 'pageform'/'autopageform' confine the maximum length of a footnote of a page to 85% of the effective text field (effective text field: page length minus length of #head# or #bottom# lines). If a footnote requires more space, 'pageform'/'autopageform' terminate page formatting with an error message.

5.3.8.1. Numbering footnotes

Similar text parts such as propositions, examples, footnotes etc. are normally numbered consecutively. As their exact number cannot be known when writing a longer text, 'pageform'/'autopageform' take care of the numbering.

The #count# directive instructs 'pageform'/'autopageform' to increment an internal counter (starting with the value 0) and to insert this value into the text instead of the #count# directive.

```
#count#
```

inserts the value 1 instead of the directive. Every further #count# directive increases the internal counter and the counter value is inserted again:

```
#count#
```

inserts the value 2 etc. Thus, it is possible to number any sections of texts (chapters, mathematical equations etc.) consecutively without worrying about the numbering when writing and changing the text.

Note: If 'lineform' encounters a #count# directive, the line is calculated as if three digits were in the text instead of the directive.

The #value# directive can be used to reinsert the *last* calculated count value. This is especially useful for footnotes.

```
text ..... (#count#)
#foot#
(#value#) text of the footnote
#end#
text .....
```

The result would look as follows:

text ..... (3)
text .....
.....

(3) text of the footnote

Note that in this case the #value# directive must follow the #count# directive without a further #count# directive in between. This is because – as already mentioned above – the #value# directive always inserts the last #count# value.

This can be avoided by adding a TEXT parameter to the #count# and #value# directives which serves as a tag.

```
#count ("rem1")#
```

#count ("rem1")# has the same effect as #count# without a parameter and inserts the value 4 here in our chapter. In addition to the consecutively counted value (running numbering of footnotes) 'pageform'/'autopageform' store a value which can be called again at any place within the text by #value ("rem1")# , for example, if another footnote should be referred to.

```
#count##count#
#value("reml")#
```

The first two #count# directives produce – in our chapter – the values 5 or 6. The #value# directive, however, inserts the stored value 4.

This is especially useful when a footnote is to be referred to in the text.

Example:

You are writing a multi-page brochure about a new product. On page 5 you want to refer to a footnote which is on another page. Then you insert 'see also annotation (#value ("delivery date")#)' in your text and continue writing. 'pageform'/'autopageform' later insert the corresponding number for the reference.

```
The word processing course is a training program for be-
ginners.

#(count)("delivery date")#)
#foot#
(#(value)("delivery date")#)
The word processing course will be available from August.
#end#
The program is based on recent findings in educational re-
search studies. The course includes textbook, workbook and six
cassettes.
```

If the number for the reference or for the footnote is to be put up, the #u# and #e# directives have to be added.

```
#u# (#value("delivery date")#)#e#
```

In the printed brochure it would be as follows (after lineform):

The word processing course is a learning program for beginners.[1] The program is based on recent findings in educational research studies. The course contains textbook, workbook and six cassettes.

Sometimes it may be necessary (as with page numbers) to reset the internal counter.

```
#setcount (13)##count#
```

produces the value 13.

[1] The word processing course will be available from August.

5.3.9. Cross references

Cross references, which will be inserted into the print file by 'pageform'/'autopageform', can be written using the #topage# and #goalpage# directives.

Cross references refer to other passages in the text and are normally only used in longer texts. In order to spare the reader the troublesome search for the text passage, the page number is usually given. Usually the page number is unknown before the text is finished . Here, 'pageform'/'autopageform' may also be helpful. The #topage#- directive references another page in the text where a #goalpage# directive is to be inserted. Instead of the #topage# directive the page number of the page on which #goalpage# is placed is inserted. In order that each #topage# is able to find its corresponding #goalpage#, a TEXT parameter is added to both directives.

```
... see also page #topage("function keys")#  ...
```

On another page, there is .

```
...    #goalpage("function keys")#
```

The corresponding page number will be inserted after 'page'.

It is possible to refer to the same (goal) page several times. You must only make sure to use the same tag (TEXT parameter) in each case. Note as well that the #goal-page# directives must be in lines that are actually printed. Do not place them in the first lines of a page or a text that contain layout directives.

The number of cross references is not to exceed 300.

5.3.10. Combination of tables, footnotes and headers or bottom lines

Tables can be placed in footnotes, #head# or #bottom# areas.

```
#head#
#lpos(0.0)##cpos(5.0)##rpos(11.0)#
#table#
Corrections   EUMEL User Manual    page 007

#table end#
#end#
```

The above entries write the following text at the beginning of each page:

Corrections EUMEL User Manual page 007

The table should therefore be completely contained in the above mentioned areas.

5.3.11. Formatting columns

The #columns# directive is used to format a text in columns ("newspaper print").

The #columns# directive instructs 'pageform'/'autopageform' to format the text into columns. The column width is set with the #limit# directive.

```
#limit (18.0)#
...
#columns (2, 2.0)#
#limit (8.0)#
...
```

The user starts writing with a line width of 18 cm. Then two-column print is requested with the #columns# directive (spacing between the columns is to be 2 cm). Thus the #limit# directive which is valid for both columns has to be set to 8 cm.

'pageform' will perform interactive column formatting as usual. The end of the column will appear on the screen with the column number displayed. Footnotes are arranged column by column and must therefore have the same line width as the rest of the columns.

'pageform'/'autopageform' generate the columns in the print file one after the other. The following example shows a section of the print file with header and bottom lines in two-column printing:

```
head lines
xx
xx
xx
bottom lines
#page##------- end page 1 column 1 ----#
xx
xx
xx
#page##------- end page 1 column 2 ----#
```

The second column appears without headers or bottom lines which, however, are taken into account in the calculation. Note that headers or bottom lines may exceed the columns. This is achieved by inserting the respective #limit# directives in these areas.

Most printers place the second column next to the first one in the printout. With a few printers the columns have to be glued side by side.

In column-by-column formatting all directives work as usual. The #free# directive, for instance, keeps space free in a column. However, the #page# directive works differently. It sets a column end. The #page# directive with a parameter (specifying the page number of the following page) sets a page break.

The #columns end# directive ends the column-by-column formatting. It has the same effect as a #page# directive.

Headings (or text blocks) spanning several columns are only possible on the first page directly after the #columns# directive.

```
#page#
#limit (10.0)#
Headings (or text blocks) spanning several columns are only
possible on the first page directly after the
#columns# directive.

#columns (2,2.0)#
#limit (4.0)#
The first column is to contain only a few pages. The premature
end of a column is reached with the #page# directive.
#page#
In the second column the user can continue writing his text.
.....................................
.....................................
.....................................
.....................................
.....................................
#columns end#
```

Printout (after 'lineform'):

Headings (or text blocks) spanning several columns are only possible on the first page directly after the #columns# directive.

The first column is to contain only a few pages. The premature end of a column is reached with the #page# directive.

In the second column the user can continue writing his text.
.....
.....
.....

The lines for the two-column heading are taken into account. However, this is only valid directly after the #columns# directive. If this effect is required again, end the column with #columns end#, write the wide heading and turn on the #columns# directive again (each time setting the correct #limit#).

5.4. Index

5.4.1. Generating indexes and/or tables of contents

The 'index' program is able to generate indexes or tables of contents. Several indexes can be merged with 'index merge'.

By calling

```
enter command :
index ("file name.p")
```

words marked by index directives are stored in files, called index files.

Words which are to go into an index have to be marked in the print file for 'index' with directives. Such index directives are ignored by the other text processing programs ('lineform', 'pageform', EUMEL printer). The user can therefore decide which words are to be included into an index while writing with the Editor.

Such word lists are generally called indexes in the EUMEL system. 'index' can also be used to draw up a table of contents and/or a list of all diagrams or to check bibliographic references.

After one or several index files have been created from a print file, the index files are sorted alphabetically upon request. Of course, tables of contents should not be sorted. After sorting, identical entries are automatically put together and listed as a single item entry with the corresponding page numbers.

Practical tip:
If you do not want identical entries to be listed as a single entry for your sort, reject the sort when prompted. Then the index file can be sorted with 'lex sort ("index file name")'. This procedure will preserve identical entries.

With the program

```
enter command :
index merge ("file name.i1", "file name.i2")
```

two indexes generated by 'index' can be merged and sorted again upon request.

5.4.1.1. Marking words for 'index'

The words to be included in an index are marked by #ib# und #ie#.

Since an index should contain the page number in addition to the actual word entry, the 'index' program only works from a print file, i.e. an output file of 'pageform'/-'autopageform'. The index words are collected in index files. The names of the index files consist of the name of the file to be processed along with an '.i' and the number of the index.

```
... A feature of the #ib(1)#EUMEL
    system#ie(1)# is described here ...
```

The words marked by the #ib# and #ie# directives are written into the first index file together with the corresponding page number.

The entries in an index file are separated from the page numbers by at least three dots. If this is not desired, the dots can easily be removed using the Editor.

Up to ten different index files can be created, e.g. words marked by

```
#ib (1)# and #ie (1)#
```

are put into the index file with the number 1, words marked by

```
#ib (9)# and #ie (9)#
```

are put in the index file with the number 9. If only one index is to be created, the #ib# and #ie# directives may be used without parameters; this is identical with #ib (1)# and #ie (1)#.

The words marked by #ib# and #ie# directives can also go beyond line borders (with hyphenation).

```
.... #ib#many index direc-
tives#ie# ...
```

'index' contracts divided words (here: 'many index directives'). If words are to be included in different index files, it is also possible to "nest" the #ib# and #ie# directives. This is especially useful for chapter headings.

```
#ib(9)#A directive: the '#ib#limit#ie#' directive#ie(9)#
```

In this example, the table of contents is put into the index file 9, while the "general" index is collected in index file 1.

5.4.1.2. Producing secondary entries

Any text may be added to the page number of any entry.

Example:

```
EUMEL-System ... 27ff.
Monitor ........ 13(Def.)
```

This is achieved by using another form of the #ib# directive:

```
... the #ib(1,"(chap.4)")#EUMEL Editor#ie# is well
 suited for editing texts ...
```

generates the following entry:

Printout:

EUMEL Editor ... 1(chap.4)

Additional texts can be added to an entry in order to form subentries:

Printout:

EUMEL system 27

EUMEL system, complex 29

This is also achieved by another form of the #ie# directive:

```
... the #ib#EUMEL system#ie(1,", user friendly")# is a really
user friendly system ...
```

creates the following entry:

Printout:

EUMEL system, user friendly 28

After setting up an index file, the entries may be sorted – the system issues a prompt. Sorting is carried out alphabetically in accordance with DIN 5007, sections 1 and 3.2 (umlauts will be sorted "correctly").

As already mentioned, 'index' can be used for many purposes:

a) Generating indexes:
   As described above.

b) Generating tables of contents:
   Enclose chapter headings in user-defined index directives and process them as described using 'index'.

```
#ib(9)#6.1. Printing a file#ie(9)#
```

Thus one is sure that the page numbers and chapter headings in the table of contents are correct.

c) Generating lists of diagrams:
   Process diagram headings in the same way as chapter headings.

d) Checking references for completeness:
   Enclose all references in separate index directives.

```
#ib(0)#/Smith82/#ie(0)#)
```

and then check the references with the aid of this index file. Thus you can be sure that all references in the text are also included in the reference list.

5.4.1.3. Merging index files

The 'index merge' program is used to "merge" one index file with another.

Thus it is possible to generate an index consisting of several files by processing the print files of these files using 'index' and subsequently combining the resulting index files using 'index merge'. Index files can be processed using the Editor or 'lineform' and/or 'pageform'/'autopageform' and subsequently be printed.

```
enter command:
index merge ("1.chapter.il", "2.chapter.il")
```

Here the index file of '1.chapter' is merged with the index file of '2.chapter' and sorted, if desired.

5.5. Outline

5.5.1. Generating an outline or a summary

The 'outline' program draws up a summary of all (chapter) headings and keywords of a text provided that they are marked by index directives.

Sometimes keywords or the table of contents are to be extracted from the text without undergoing 'pageform' before. This is especially useful if

- keywords are to be checked for their correctness and completeness;
- the sequence of chapters is to be checked;
- an outline of chapter headings and keywords is to be made;
- a text is to be checked for logical combination.

In such cases the 'outline' program is useful which can be called with the Monitor command

```
enter command :
outline ("file name")
```

'outline' works similar to 'index' by writing all text sections marked by #ib# and #ie# in a file with the addition 'outline'. In contrast to 'index' the entry file does not have to be a print file ('.p' addition).

The 'outline' program asks first what index number the table of contents bears. This is necessary because in the 'outline' file the chapter headings are made to stand out against the keywords (indents).

Entry file ("file name"):

```
...
#ib(9)#1. Chapter#ie(9)#
...
...#ib#Keyword 1#ie#
#ib#Keyword 2#ie#...

#ib(9)#1.1. Chapter#ie(9)#
...
#ib#Keyword 3#ie#
etc...
```

Printout of the generated file ("file name.outline"):

1. Chapter
   Keyword 1
   Keyword 2
   1.1. Chapter
   Keyword 3

In this example all indexes with the exception of the chapter heading are listed in a separate line and are indented compared with the chapter heading. A new chapter, if it is marked decimally, will be indented compared with a chapter of higher rank.

5.6. Print

The EUMEL printer which is addressed with the 'print' command is a software interface to a connected printer. This section explains how to print a file using the EUMEL printer and the special directives controlling the printer.

The performance characteristics of printers (e.g. type fonts and modifications) differ depending on the hardware. These print features are initiated by entering special character strings which are interpreted differently by the printer manufacturers.

In order to enable the EUMEL system to address different printers in the same manner, a software interface has been created which is called the EUMEL printer. The EUMEL printer accepts a file as input and has it printed in a suitable way. Furthermore, the EUMEL printer observes the text cosmetics directives. The form of directives for the text cosmetics and for the EUMEL printer is identical.

5.6.1. Printing a file

The 'print' command can be used to pass a file to the EUMEL printer for printing.

```
enter command :
print ("file name")
```

Normally a "spooler" is installed in the EUMEL system (multi-user) so that you can continue working immediately. In this case the EUMEL printer works parallel to the other activities of the user.

It is also possible to pass several files to the EUMEL printer by using a thesaurus. (see p. 3-36ff)

Example:

```
enter command :
print(SOME all)
```

5.6.2. Directives for the EUMEL printer

A text (a file) can also be printed by the printer without directives, e.g. for provisional printouts. In this case the printer has reasonable defaults. For a "normal" text, you do not need to insert special printer directives into the text to be printed since the directives for the text cosmetics are sufficient to control the printer. Only if special features are required, such as right justification or placing the printed text at a certain position, printer directives are necessary.

If features are requested from the printer which are not available in its hardware, the EUMEL printer tries to perform the closest equivalent feature. If, for example, a non-existent type font is requested, the text is printed using the standard type font of the respective installation. This makes it possible to print a text, which is actually intended for another printer, on a printer that does not know the type font requested.

As already mentioned, the EUMEL printer observes the same directives as the text cosmetics programs but there are a few directives that are only implemented for the printer. A #type# directive which requests a certain type font, for example, is passed on by the EUMEL printer as a directive sequence to the connected hardware printer, provided the type font exists on the printer. Details on how to write the directives are given in the description of text cosmetics.

Directives are not printed. If a line only consists of directives, this line is not printed by the EUMEL printer. In contrast to the programs of the text cosmetics, directives that are unknown or erroneous are "swallowed" by the EUMEL printer without an error message being given.

Apart from the "normal" directives which are only enclosed in '#' characters, there is another form:

comment directives:

These are enclosed in "#-" and "-#" characters. Such directives are ignored.

```
.........................
text.....................
.........................
comment directives are
ignored during printing.
#------------------ end of page 1 -------------------#
```

The last line does not appear in the printed text.

5.6.3. Right justification

5.6.3.1. Justification

The #block# directive turns on right justification during printing.

If the

```
#block#
```

directive is inserted into the text (usually at the beginning of a file), the printer will print all lines after this point that are not marked by a paragraph characteristic right justified. This means that, by increasing the spacing between words, all lines end at the same position (right justification). With low-priced printers, this is only possible by inserting whole blanks between the words which often makes the text more difficult to read. With high-quality printers, however, right justification is accomplished by inserting smaller spaces between words.

The text of a line is widened by enlarging the word gaps to the line width set by the #limit# directive.

a) Not to be enlarged:

- paragraph lines;
- the text up to the last multiple blank;
- leading blanks (indent);
- a blank after a list (cf. b);
- protected blanks.

b) Lists only exist after a line marked by a paragraph character:

- dash (hyphen and blank at the beginning of a line);
- colon at the end of the first word (position < 20);
- closing paranthesis or full stop at the end of the first word (position < 7), e.g. 1) or 1.

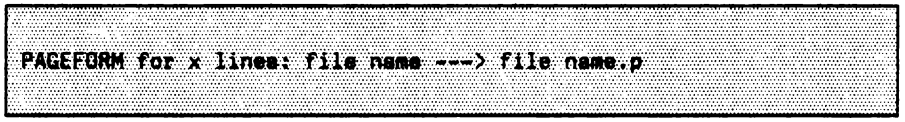
5.6.3.2. Page justification

The #pageblock# directive instructs the printer to carry out a page justification (similar to the #block# directive for right justification).

Due to automatic or interactive page formatting or footnote makeup by 'pageform'/-'autopageform', lines often remain empty at the end of a page. This can be avoided by using the #pageblock# directive. It instructs the printer to insert space (technical term: lead) between the lines so that the last lines on all pages end at the same height. As with the right justification, here the quality of the printout also depends on the efficiency of the printer being used.

Note that some publishing companies do not wish pages to be processed in this way because lines may "shine through" when being printed, if the paper used is too thin. Therefore reading these pages becomes more difficult.

If the #pageblock# directive has been given, the page break can also be placed beyond the arithmetic page end in 'pageform'. In this case the lines are compressed by the printer.



5.6.4. Moving the text field

The #start# directive makes is possible to place the text field at another position on the paper when printing.

The EUMEL printer places the text field automatically so that a sufficient margin remains. The efficiency of this presetting obviously depends on the printer and on the installation. The automatic setting can be changed with the #start# directive.

```
#start (1.0, 2.0)#
```

determines the upper left corner of the text field (1 cm from the left edge, 2 cm from the upper edge). The standard presetting is #start (2.54, 2.35)#. The #start# directive can only be given once per page.

5.6.5. Centering

With the #center# directive a text can be printed in the middle of a line.

The #center# directive centers the words of a paragraph line.

```
#center#This line will be printed centered.
```

Printout:

This line will be printed centered.

5.6.6. Printing right justified

With the #right# directive part of a paragraph line can be printed right justified.

The #right# directive prompts the following text to be printed right justified.

```
#head#
#center#This line will be centered#right#%

#end#
```

In the above example the page number will be printed right justified.

Note that the #center# and #right# directives can be used together. However, both directives only work if they stand in a line marked by a paragraph mark.

5.6.7. Printing characters on top of each other

With the #b# directive two characters can be printed on top of each other.

The #b# directive prompts two characters following one another and which are joined by the #b# directive to be printed on top of each other.

```
... 0#b#/ ...
```

Printout:

... 0 ...

The '/' character is printed on top of the '0' character. 'lineform'/'autoform' consider only one character for calculating the line. Note that no blank is to stand directly in front of or behind the #b# directive.

5.7. Text cosmetics macros

Macros are used to abbreviate recurring sections of texts and/or directives.

By 'macro' a "large" directive is understood which consists of many small ones and which can be called with the aid of the macro name.

Text cosmetics macros are used for:

- constantly recurring text sections;
- constantly recurring directive sequences;
- manuscripts whose final form is not yet known when they are created or which are to be changed later;
- sequences of direct printer directives which produce a certain performance.

Macros are defined using the Editor. Then the macro file is loaded. From this point onwards 'lineform'/'autoform' and 'pageform'/'autopageform' "know" the macros, i.e. the text lines and/or directives "hidden" behind the macro name.

'lineform'/'autoform' observe the directives contained in the macros. However, they do not appear in the file. 'pageform'/'autopageform' will later insert them into the print file.

5.7.1. An example of macros

In the following a simple example of a letter head is shown.

Supposing the EUMEL system is used for writing business correspondence. A printer is available to you with which letter heads can be generated. For the letter head a macro #head# is written into a file "macro definitions":

```
#*head#
#type("bold and large")#company name
#type("bold")#software products
#type("small")#street
town
#type ("normal")#
#*macro end#
```

The name of the macro is #head#. Note that a macro definition has to start with the name of the macro. The macro name must be marked with a * in order to distinguish it from "normal" text directives. Each macro must end with a #*macro end# directive. Several macros can be written sequentially into the file.

Now, the macro thus defined has to be loaded:

```
enter command :
load macros ("macro definitions")
```

The "loaded" macros can be displayed in a notebook in order to check them:

```
enter command :
list macros                                    .
```

From now on a new directive (with the name #head#) will be available which can print a letter head in every letter. The following letter can now be written:

```
#head#

Dear Sir ....

etc.
```

Note that the macro appears in the text as a directive without the *. Activating a macro which, for instance, is in a file to be processed by 'lineform' does not differ from a "normal" text directive.

After formatting the letter line by line with 'lineform', the formatted file is checked. Nothing has changed yet. The new directive #head# is unchanged in the file. 'lineform' observes all directives and text lines of a macro but does not put them into the file. 'lineform', however, cannot recognize #type# and #limit# directives of a macro if the macro is standing at the beginning of a file and in whose definition the directives are correctly specified at the very beginning. 'lineform', nevertheless, requests 'type' and 'limit' at the beginning. This can be avoided by ignoring the request by pressing 'CR'.

Now the file containing the letter is formatted with 'pageform'/'autopageform'. The #head# directive has now disappeared in the print file. It has been replaced by the lines of the macro body. 'pageform'/'autopageform' insert the lines of the macro into the print file:

```
#type("bold and large")#company name
#type("bold")#software products
#type("small")#street
town
#type ("normal")#


Dear Sir ...
etc.
```

Note:
Macros with identical names, but different parameters are not allowed. It is not permitted to call macros within a macro definition either.

Note also that macro texts are used in the same way as they were loaded with 'load macros'.

```
#*beginning of text#
#limit(11.0)#
#block#
#pageblock#
#type("trium8")#
#*macro end#
```

If the CR key (paragraph) is pressed in the macro file after each line, then a paragraph is made after each #...#, which is desirable for chapter headings. This, however, is not the case with smaller directives after which a paragraph would then appear in the middle of a sentence. Macros for such applications should be stored without paragraphs. Note further that, due to program – technical reasons, a #foot# or the closing #end# directive of a footnote cannot be contained in a macro.

5.7.2. An example with macro parameters

Macro parameters make it possible to generate recurring text sections which only differ slightly from each other.

You realize now that your macro could be improved further. You want to include the date in the letter head. Therefore, you edit your macro file as follows (note the '$' characters):

```
#*head ($1)#
#type("large")#company name
#type("bold")#software products
#type("small")#street
town
#type ("normal")#

                                        town,  $1
#*macro end#
```

Thus, you have added a parameter to the #head# macro: '$1'; the parameters are numbered. A second parameter would be called '$2' etc.

If you write a letter, you have to write the #head# directive into a letter with the current date:

```
#head ("20.8.1986")#
```

'pageform'/'autopageform' will insert the date entered directly after 'town, ' into the letter head (in the print file). Note that all parameters in a macro directive have to be in quotation marks (also numbers).

5.7.3. Macros for manuscripts

The following shows how macros can be used to formulate directives stating what a text is about and not, in which format it is to be printed.

When writing manuscripts for articles, books and manuals authors often do not know in which format the manuscript will be printed. For this purpose it is also helpful to use macros.

```
#*chapter begin ($1)#
#free (2.0)#
#type ("large")##ib (9)#$1#ie (9)##type ("normal")#

#*macro end#
```

In this example, a macro is defined for the beginning of a chapter. Here, two centimeters of space should remain between two chapters, the chapter heading (as a parameter) is printed in a large type font. In additon, the heading is included in the ninth index to create a table of contents. A blank line is inserted after the heading before the actual text begins.

The user of this macro writes, for example, the following directive:

```
#chapter begin ("An example of manuscripts")#
```

Note that the chapter heading must not be longer than a text line. This is because 'lineform'/'autoform' process the line but do not insert it into the text. Thus 'pageform'/'autopageform' insert the text line unchanged (not split up).

Macros can be defined for most text structures. In this case, typists normally do not need to know that many text directives, but only a few simple ones.

The macro directives can be changed at any time in order to adapt them to changing requirements, e.g. when a publisher bindingly requests a specific text format. In this case, only the macro definitions have to be changed, and not all text files.

A further advantage of such a procedure is that the macro directives indicate ***what*** a certain text structure is and not ***how*** the structure is to be treated.

Note:
If necessary, #limit#, #type# and #linefeed# specifications should be inserted into a macro definition in order to make the macros independent of the call location. Likewise, the file should be processed with 'lineform' beforehand if hyphenations are requested.

5.8. Text cosmetics for specialists

In this section commands and directives are introduced which are, as a rule, only used for special purposes.

5.8.1. Switch instructions for head and bottom areas

With the text cosmetics directives

```
#head off#
#bottom off#
```

the generation of header and bottom lines can be switched off. With

```
#head on#
#bottom on#
```

they can be generated again. Note that these directives are already being taken into consideration at the position where they are placed in the text, i.e. these directives are already valid for the page on which they are during the 'pageform' processing. If the header lines are to be switched off for one page, then the #head off# directive should be given at this point. In order to switch on the header lines again for the next page, the #head on# directive should be placed at a position from which one can be sure that it will get onto the following page (if in doubt after a #page# directive).

5.8.1.1. Switching off head and bottom areas

With 'first head (...)' or 'last bottom (...)' head or bottom areas on the first (last) page can be switched off or on again.

Sometimes it is necessary to avoid the generation of 'head' lines on the first page (e.g. because a letter head appears there) and/or the generation of 'bottom' lines on the last page (because no following page exists). With the Monitor command

```
enter command :
first head (FALSE)
```

the generation of 'head' lines during 'pageform' can be switched off for the first page of each print file. The generation is turned off until it is switched on again with

```
enter command :
first head (TRUE)
```

The same applies analogously to 'bottom' lines on the last page: switching off and on with

```
enter command :
last bottom (FALSE)
```

or

```
enter command :
last bottom (TRUE)
```

5.8.2. Marking lines of text

With the directive

```
#mark("mark character left","mark character right")#
```

a text passage can be marked with texts at the margins (outside the text field) as, for example, in the following with the directive

```
#mark ("> ", " <")#
```

> Here the first parameter applies to the left margin and the second one for the right margin. Note that sufficient space has to be given between the mark and the margin. <
>
> The mark is especially interesting for manuals where changes are made to stand out against the last version. The mark character is printed next to the left and right margin (i.e. outside the text field limited by #start# and #limit#). For the printing of the mark the type font/the modifications are used that are valid at the position where the #mark# directive is found. The actual text itself, of course, remains untouched. <

In order to mark only one margin, a dummy parameter can be given, too.

```
#type ("pica")##mark ("", " |")##type ("normal")#
```

With the special #mark# directive

```
#mark ("", "")#
```

the mark is switched off.

If a head, bottom, footnote or table area is to be marked, the directives for switching the mark on or off should be completely contained in the respective area.

5.8.3. Counting footnotes per page

Sometimes it is necessary to count the footnotes for each page separately, i.e. for each page starting from page 1. This can be done with the text cosmetics directive

```
#count per page#
```

It switches from continuous counting to counting page by page. This directive should be placed at the beginning of a file. It cannot be switched off any more for the file in question.

5.8.4. Treatment of incorrect hyphenation: dictionary of exceptions

Incorrectly divided words can be entered into a dictionary of exceptions.

It may happen that the hyphenation program of the text cosmetics divides words incorrectly again and again. In order to avoid this, these words can be stored in a ***dictionary of exceptions***. At first the entries in the dictionary of exceptions are searched when words are to be divided. If a word is found in this dictionary, then the actual hyphenation program is not carried out any more.

As described below the exceptions have to be noted in a file and to be loaded with the Monitor command into the dictionary.

```
enter command :
load exceptions ("file name")
```

The exceptions have to be written into the file as follows:

```
au-to-crat-i-cal
ea-pe-cial-ly
room
ex-am-ple
...
```

New exceptions can be loaded at any time (again with 'load exceptions'). In this case you are asked for confirmation if the dictionary is to be overwritten.

In order to check which or how many exceptions there are in the dictionary the following command can be given:

```
enter command :
unload exceptions ("file name")
```

Then the dictionary is written into "file name". At this point further exceptions can be added as well and be loaded anew (this time, however, overwriting).

5.8.5. Changing presettings: some Monitor commands

5.8.5.1. A few or many hyphenations: setting the hyphenation frequency

With the 'hyphenation width' command the position at which words are offered for hyphenation is determined. Hyphenation width can be set from 4 to 20 percent of the line width.

Many hyphenated words in a text make it difficult to read. Without hyphenations, the right margin is either "uneven" or, with right justification, many spaces must be inserted between the words. With the Monitor command

```
enter command :
hyphenation width (percentage)
```

immediately before calling 'autoform' or 'lineform' the position at which the hyphenation is to start can be set. The brackets contain an integer which represents the percent of the line width. Minimum is 4, maximum is 20 percent. For example, 'hyphenation width (5)' (7 is default) means for a line width of 80 characters that hyphenation is to be performed if more than 4 blanks occur without hyphenation. This is valid for equidistant type fonts. With a proportional spacing the percent figure refers to empty space and not blanks. If 20 is specified very few words will be offered for hyphenation, i.e. the larger the percent specification the fewer words will be offered for hyphenation. Thus the specification of the hyphenation frequency determines at which position a word is to be checked for hyphenation. On the other hand, this specification also determines how many spaces must be inserted between words in order to achieve right justification.

5.8.5.2. Setting the number of blank lines before footnotes

'number empty lines before foot' sets the number of blank lines before footnotes.

The number of blank lines before footnotes (one blank line is default) can be set with the 'number empty lines before foot' command.

```
enter command :
number empty lines before foot (3)
```

sets three blank lines before the footnote block. Note that this setting is valid until the Monitor command is given again.

5.9. Outline of directives and commands of the EUMEL text cosmetics

First the most frequently used commands/directives are described. Then (detached by a line) commands/directives are listed which are required less often.

Commands

Commands are given in the Monitor ('enter command :').

command	function
lineform ("x")	line formatting with interactive hyphenation
autoform ("x")	line formatting with automatic hyphenation
pageform ("x")	interactive page formatting taking footnotes, header and bottom lines, page numbering, cross references etc. into account. Generates a print file (addition '.p').
autopageform ("x")	same as pageform, the page breaks, however, are placed automatically.
print ("x")	printing a file
print ("x.p")	printing a file processed by 'pageform'
- - - - - - - - - - - -	- - - - - - - - - - - - - - - - - - - -
index ("x.p")	generates an index and/or a table of contents.
index merge ("a.i1","b.i1")	merging index files
outline ("x")	generates an outline of chapter headings and key words

hyphenation width (int)	determines the hyphenation frequency
load macros ("x")	loads macros
list macros	displays loaded macros
load exceptions ("x")	loads words, which are not correctly separated by the hyphenation program, into a memory of exceptions
unload exceptions ("x")	unloads the words from the memory of exceptions into the file specified
first head (false)	switches off the header lines on the first page
first head (true)	switches the header lines on the first page on again.
last bottom (false)	switches off the bottom lines on the last page
last bottom (true)	switches the bottom lines on the last page on again
number empty lines before foot	sets the number of blank lines before a footnote

Directives

Directives are written into the file. Each directive has to be enclosed in directive characters. Possible parameters are (written in brackets):

'int' stands for an integer: 17, 1, 311;

'real' stands for a number with decimal point (specification usually in cm): 0.5, 1.25;

'text' stands for a character string entry. Has to be written in quotation marks: "%", "my file".

directive	function
type (text)	setting the type font: #type("trium8")#
limit (real)	setting the line width: #limit (16.0)#
on (text)	switching on modifications: #on("bold")#. Possible are: b(bold), r(everse), i(talic), u(nderline).
off (text)	switching modifications off (cf. 'on')
block	switching on right justification
head	defining header lines (for pages with even/odd
(or headeven/headodd)	page numbers)
... %	stand – in for page number
end	end of header lines (pageform)
bottom	as above, however, for bottom lines
(or bottomeven/bottomodd)	
...	
end	end of bottom lines
pagenr (text, int)	setting the page number or introducing additional page character from the next page: #pagenr ("%", 17)#
foot	beginning of footnotes
...	
end	end of footnotes
free (real)	keeping space free (in cm): #free (1.27)#
page	new page: #page#

page (int)	new page with page number 17: #page (17)#
linefeed (real)	changing line height in relation to the set type font: #linefeed (1.25)#
pagelength (real)	setting page length (from next page in cm): #pagelength (24.0)#
center	centering the following text of the line
right	printing the following text of the line right justified
u ... e	(stands for 'up') writing exponent: #u#123#e#
d ... e	(stands for 'down') writing index
start (real, real)	setting text field (upper left corner): #start (1.0,2.0)#

---

b	printing two characters on top of each other
bottom off	switching off bottom lines
bottom on	switching on bottom lines
bpos (real, real)	the text between the specified table positions is printed right justified.
clearpos	clearing all table positions
clearpos (real)	clearing the specified table position
columns (int, real)	formatting columns with space in between: #columns (3, 1.0)#, 3 columns with one cm space in between
columnsend	ending the column formatting
count	internal counter for footnotes is set (pageform)
count (text)	as above, but the value of the internal counter is noted: #count ("new number")#.
count per page	internal counter starts on each page with 1.
cpos (real)	centering table position
dpos (real, text)	centering table position round the specified text, mostly decimal characters: #dpos (13.0, ".")#
fillchar (text)	space between table positions is filled with the specified text when printed. Note that the switching off of the fill characters is done by 'niltext'.
goalpage (text)	position to which the above directive refers: #goalpage ("chapter 1")#

head off	switching off header line(s)
head on	switching on header line(s)
ib	marking the beginning of a key word or a chapter heading (puts it into an index file with the addition '.i1'): #ib#a key word or a chapter heading#ie#
ib (int)	as above, however, the key word is put into specified index file.
ib (int, text)	as above, however, the entry of the index file is completed with supplemented text for the page number.
ie	end of an index
ie (int)	as above (int specification must correspond to the one in the ib directive)
ie (int, text)	as above, however, the text is added to the marked index.
lpos (real)	left justified table position
mark (text, text)	switching the marking next to the text field on and off
pageblock	switching on vertical justification. If it is switched on, formatting can go beyond the (arithmetic) page break with 'pageform'.
rpos (real)	right justified table position
setcount (int)	setting counter value: #setcount (17)#
table	beginning of a table
...	
table end	end of a table
topage (text)	page cross reference (the page number that is referred to is inserted): #topage ("chapter 1")#
value	inserting the last count value
value (text)	as above, however, an entered counter value is inserted: #value ("entry")#.

5.10. Possible errors and how to remedy them

What can you do if

certain directives that are valid for the entire text only become effective from the second page onwards?

Text cosmetics directives which are to be valid from the first page onwards have to be written first of all, i.e. in the first line of a file. This includes, among other things, 'pagelength', 'start', 'block', 'pageblock' etc., which are to be placed before the #head# or #bottom# directives so that they are also valid for them.

the cursor cannot be moved any more?

One possibility is that you pressed the 'STOP' key (= 'CTRL a' simultaneously, i.e. stopping the screen display) by mistake. In such a case, press the 'PROCEED' key ('CTRL c' simultaneously, i.e. continue screen display). All key strokes that have been carried out in the meantime are now performed.

Another possibility could be that you did not leave your file/task correctly. Try to get back onto the Monitor level with the help of the 'SV' key and 'ESC h' so that after the prompt 'enter command' you can get access to your file again.

you want to delete, duplicate or process only a section of a file with 'lineform'?

The section in question has to be marked. For deleting use the 'ESC RUBOUT' keys. The section, however, has not 'completely disappeared', but can be reproduced with 'ESC RUBIN' at the same or another position until you use the keys again.

Duplicating a text section takes place after having marked it and then pressing the key sequence 'ESC d'. Here the original text is preserved and can be duplicated as many times as desired. The duplicated text is fetched with 'ESC g' to the desired position in your file.

If you want to apply 'lineform' only to a section, you mark it and after pressing 'ESC ESC' you enter the command 'lineform'.

the last line or the last two lines of a page are printed on a separate page during printing?

a) The fonttable has still to be set.

   or

b) You set #pageblock# directive at the beginning of the text and move the page break interactively down by two lines.

   or

c) You choose a smaller type font.

the message

```
ERROR: FILE Overflow

enter command :
edit ("file name")
```

appears in your file and carrying out the command with the help of the 'CR' key only leads to an identical message (see above)?

If you do not get back into your file after having tried it several times in the above mentioned way, it is possible to 'organize' your file anew with the command

```
enter command :
reorganize ("file name")
```

in order to eliminate 'gaps' which were caused by inserting or deleting. Then the file usually requires less storage space.

When you are back in your file, it is advisable to divide the large file into several small ones. Either you divide it into two or (even better) you divide it into three parts and distribute the text on two or three files. In future you should make it a rule with extensive texts to file only a logically coherent chapter in one file. Your files should only be of such a size that there is enough space left for carrying out procedures like 'pageform' by which the size of a file is increased (sometimes considerably).

some words in your text were printed with an extremely large space in between them?

In such a case you most probably forgot to set a paragraph mark so that, because of the #block# directive, right justification was carried out which you did not require at this position.

overwriting has occurred or if 'lineform' reports an overwriting error?

Overwriting can happen if you use an especially large type font (e.g. triumb14) in bold-face printing (see also 5.2.6.). In order to avoid overwriting the number of blanks between the individual components can be increased or the #type# directive can be placed in the preceding line (not directly in front of the text concerned).

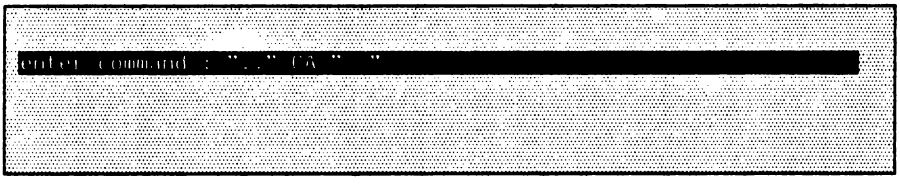
your tables (which were not embedded in #table# and #table end# directives) got mixed up after the 'lineform' procedure?

When writing tables, a paragraph mark has to be set after each line.

Check also whether there are sufficient double blanks available so as to make the columns aligned.

no fill characters are required between the text and the page number when making a table of contents or an index?

In this case press 'ESC ESC' to get into the command mode and by means of CA (Change All) change the fill characters into blanks. Do not use only one dot because otherwise the dots between the figures of the chapter numbering would be deleted. If there is an odd number of dots, it can become necessary to remove some dots later.



This procedure is very time-consuming. Therefore it is advisable to remove several dots at once (the more the better) and to replace them by an equivalent number of blanks.

PART 6: Special features

6.1. Notebook

Among other things the notebook permits the temporary storage of error messages and displays the error messages together with the processed text in the window Editor at the end of a process.

The notebook is used for collecting warnings and error messages by the 'lineform' and 'pageform' programs. When the window of the notebook is opened on the screen, it can be handled just like the usual Editor window.

If you yourself want to use the notebook when editing, press the ESC n keys at any position on the screen instead of ESC e for the window Editor. Through this command the notebook is displayed instead of a file. Thus, you save the input of a file name and can start working with the introduced Editor functions in the notebook straight away.

```
.................. handbook part 6 ................. line 55
#kapon#6.2. EUMEL character set#kapoff#
   #corner1("-5.0")#
   The EUMEL system defines a character set which guarantees that
   characters are coded identically on all machines. Thus, it is
   to transfer files or programs without conversions between
   manufacturers. The EUMEL character set is based on the ASCII
   set (DIN 66 003) with expansions.
   #box3("I","2","115.0")##off("b")##type("trium8")#

.................... notebook ..................... line 1
ERROR line 55: modification not switched on with off: b
   >>> check directive in corresponding line number
WARNING line 55: switch to same type font: trium8
   >>> therefore type font was not changed!
WARNING line 48: overwriting after >#ib(9)#6.2.< Missing blank  1
   >>> Please insert missing blanks
```

6.2. EUMEL character set

The EUMEL system defines a character set which guarantees that characters are coded identically on all machines. Thus, it is possible, for example, to transfer files or programs without conversions between EUMEL systems installed on computers of different manufacturers. The EUMEL character set is based on the ASCII character set (DIN 66 003) with expansions.

The display of the individual characters depends on the terminal. The characters listed below generally exist on all machines. An expanded character set (with mathematical, diacritic, and Greek characters) is only available on special machines and therefore is not shown here.

Examples of how to read the table:

```
                code (" ") ->  32
                code ("m") -> 109
```

	0	1	2	3	4	5	6	7	8	9
3			SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~			
13										
.										
.										
.										
20										
21					Ä	Ö	Ü	ä	ö	ü
22	k	-	#	SP						
23										
24										
25		ß								

Note:

1) SP means blank character.

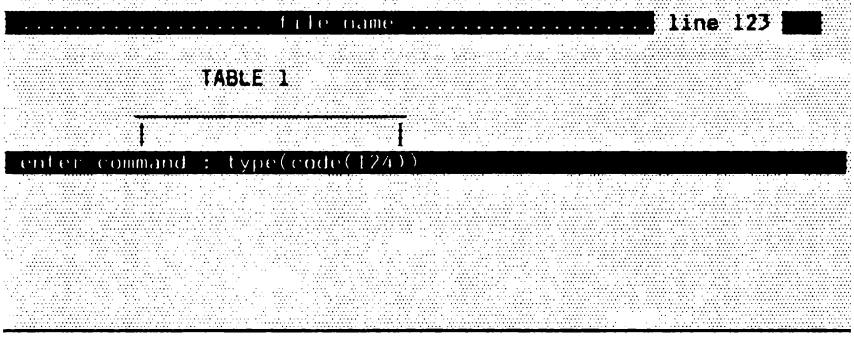
2) The characters 'k', '–' and 'SP' with the codes 220, 221, 223 are required for text cosmetics purposes (split 'k' when changing 'ck' into 'kk' (German hyphenation); split character; protected blank).

3) The character '#' (code 222) is printable, whereas the character '#' (code 35) is not (introductary character for text cosmetics and printer directives).

4) On the terminal the character SP (code 223) is displayed inversely or as an underscore character for better identification. When printed, it appears as a blank.

If you want to enter characters which are not on the keyboard, you have to make use of the code of the characters desired.

To do this, move the cursor to the position in the file in which the special character is to be printed and, after pressing ESC ESC, enter:



6.3. Sorting programs

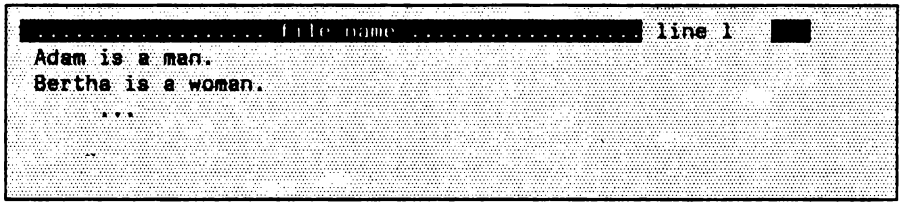
There are two different sorting programs available: 'sort' (sorting according to ASCII order) and 'lex sort' (sorting according to the German alphabet).

The sorting programs sort a file line by line.

Example:

```
.................. file name ..................  line 1
Bertha is a woman.
Adam is a man.
   ...
```

```
enter command :
sort ("file name")
```

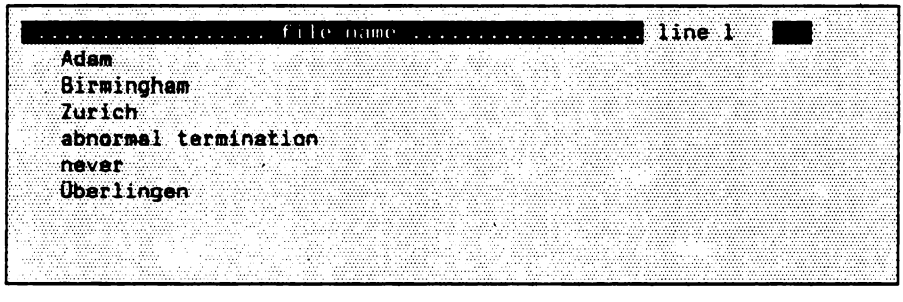


While doing so, the beginnings of a line are compared character by character until a difference occurs and then they are reordered, if necessary. If two lines (number of characters/line) which are unequal in length are compared, then you can imagine the short line to be extended by blank characters up to the length of the longer line.

The order in which the lines are sorted conforms to the ASCII character set in ascending order (cf. EUMEL character code):

```
blank character
some special characters
digits
some special characters
capital letters
some special characters
small letters
some special characters
the umlauts and ß
```

This means that the sorting is carried out as follows:



In order to achieve that capital and small letters are treated equally, you can enter the command

```
enter command :
lex sort ("file name")
```

In such a case the sorted file would look as follows:

```
                                                    line 1
abnormal termination
Adam
Birmingham
never
Überlingen
Zurich
```

Note that the umlaut 'Ü' is treated like 'Ue' (the other umlauts are treated accordingly; just as 'ß' is treated like 'ss'). Furthermore, all special characters are ignored during the sorting order.

6.4. Font tables

A font table contains specifications about the printable characters.

Setting a font table[1] takes place automatically when the printer task is created (cf. Appendix). In order to obtain the name of the font table set in this task, you enter:

```
enter command :
put(fonttable)
```

The output supplies the name of the font table set in this task.

```
enter command :
put(fonttable)
agfa9
enter command :
```

1) Font table: Description of the printable type fonts.

In order to set a new or another font table, for example, because different printers can be used, you enter the 'fonttable' command with the name of the font table desired as parameter:

```
enter command :
fonttable("name of the fonttable")
```

A more detailed description of the set font table can be obtained by the command 'list fonts':

```
enter command :
list fonts
```

Through this command you get a list in the notebook of the type fonts with specifications about the names of the type fonts available in the font table, size specifications for the type fonts etc.

```
..................... notebook ..................... line 1
FONTTABLE  :  "agfa9";
  x unit  =  160.0;
  y unit  =  160.0;

  FONT  :  "micro",  "elanlist",  "bulletin22";
    indent width   =  20;
    lead           =  7;
    font height    =  30;
    font depth     =  8;
    larger font    =  "";
    smaller font   =  "";

  FONT  :  "triuml0";
    indent width   =  31;
    lead           =  6;
    font height    =  54;
    font depth     =  15;
    larger font    =  "triuml2";
    smaller font   =  "triu8";
```

Note:

- If several names are given for one type font, you can use any of these in the #type# directive.

- Size specifications are given in 'micro units', i.e. the smallest possible units of the respective printer, and not in mm.

- Please refer to the system handbook for further information, if necessary.

6.5. Syntax of the commands

code

TEXT PROC code (INT CONST number)
Converts 'number' into a character. If the number is smaller than 32 or greater than 254, (cf. code table) you must reckon with unexpected results.

```
type(code(92))
```

INT PROC code (TEXT CONST character)
Converts 'character' into the appropriate EUMEL coding. If more than one character is specified, the result is '-1'.

```
put(code(92))
```

lex sort

PROC lex sort (TEXT CONST file)

Sorting line-by-line according to (German) lexicographic order according to DIN 5007.

```
lex sort ("telephone list")
```

PROC lex sort (TEXT CONST file, INT CONST beginning)

Same as 'lex sort', however, sorting starts at 'beginning' of each line.

```
lex sort ("list",20)
```

list fonts

PROC list fonts
Lists the fonts of the set font table into the *notebook*.

PROC list fonts (TEXT CONST fonttable name)
Lists the fonts of the specified font table into the *notebook*. The font table previously set, however, still remains valid.

```
list fonts ("fonttab.alternative")
```

sort

PROC sort (TEXT CONST file)

The procedure 'sort' sorts the file 'file' line – by – line. The sorting is carried out according to the order which is specified by the EUMEL character code. Lines, for example, which start with digits are placed before lines starting with letters. Lines starting with capital letters are placed before lines starting with small letters. In addition to that, the umlauts and "ß" are placed after all the other letters.

```
sort ("list")
```

PROC sort (TEXT CONST file, INT CONST beginning)

Sorts a file like the procedure above, however, during sorting not the beginning of a line is taken into account but the position 'beginning'.

```
sort ("list",10)
```

type

PROC type (TEXT CONST character string)

Inserts 'character string' into the current position of the edited file. Particularly useful in connection with the procedure 'code' in order to get characters into a text which are not contained on the keyboard.

```
type(code(200))
```

Appendix

Structure and installation

This installation description serves only as an example and is not to replace the manual for the unit you are using. Above all, the possibility of partitioning described here is not standard!

General information on the EUMEL operating system

For a better understanding of the installation procedure the structure of the EUMEL operating system is to be explained here (EUMEL experts may leave out this part):

The system essentially consists of the following components:

- SHard (*S*oftware – *Hard*ware Interface)
- system kernel (EUMEL 0)
- system parts based upon it

The **SHard** is the hardware-dependent part of the operating system. This part is responsible for the flow of information between the virtual EUMEL 0 machine and the individual hardware components (keyboard, diskette drive, interfaces etc.).

The **system kernel** (also referred to as EUMEL 0 or Urlader) is the processor-dependent part of the system. Essentially, it determines the performance of the operating system by defining the instruction set of EUMEL 0 for the ELAN compiler. This instruction set of EUMEL 0 is mapped onto the actual instruction set of the processor of the individual machine.

Those **parts of the system based on the kernel** (EUMEL 0) are hardware- and processor-independent. They contain the ELAN compiler and all tasks, texts, inserted ELAN packets, named and unnamed data spaces of a EUMEL system. These system parts together with the system kernel EUMEL 0 form the **EUMEL background**, i.e. EUMEL-0 is part of the EUMEL background. Depending on the type of computer, diskette and hard-disk are supported as background storage medium for EUMEL at present.

The term EUMEL background results from the concept of the virtual memory. In this concept the RAM area of the hardware is only used as a buffer area according to the demand-paging concept, with the exception of the resident system parts (SHard and EUMEL 0). So the user is independent of the actual size of the RAM in respect to his programs and data. The RAM size only determines the process speed (performance) of the system, i.e. the larger the RAM area of your computer the faster the EUMEL system works.

EUMEL 0 is contained in the first background diskette. Therefore the loading of EUMEL 0 and the rest of the EUMEL background can be carried out in one operation.

The delivered background has not been configurated yet and solely provides the performance of the system described in the manual. It basically consists of inserted ELAN packets which determine the performance capacity of the system (single or multi-user, with or without word processing). If the background is to be found on several diskettes (multi-user backgrounds), then they are continuously numbered. The first background diskette has the number 0 in order to remind the user that on this diskette the system kernel EUMEL 0 is placed as well.

Of course, a background can also be the system backup of a larger system with several megabytes, for example. If at any later stage you want to replace your present system by another one (overwriting the background), then you interrupt the system during the booting in the memory test by pressing a key. After that you load the new system by selecting 2 'load new background from archive' in the start dialog.

NOTE: As a result all data of the old system is irretrievably lost! (cf. also 'loading a EUMEL background')

In order to make the terms 'system start' and 'system shutdown' of the EUMEL system clearer they shall be explained here:

System start :

When a EUMEL system is booted, and this is also valid for smaller diskette systems like the 'EUMELgenerator' (cf. page A – 4), first the SHard is loaded; for this a corresponding message appears on the screen. The SHard now tries to load the system kernel from the archive medium (usually a diskette). If no corresponding diskette is fed in, then it tries to load EUMEL 0 from the background medium (hard disk).

After that EUMEL 0 becomes active; specifications on the available channels, RAM and background memory size appear on the screen. Then EUMEL carries out a memory test what can be seen from the fact that a series of stars (*) is written onto the screen. If meanwhile any key is pressed, then you get into the 'start dialog' after the memory test.

After the memory test or the selection of 1 'system start' in the start dialog, the background is activated; how long this will take depends on the size of the system and the kind of background medium.

System shutdown :

Before switching off the computer, each EUMEL system should be shut down correctly. This is done by the command 'shutup' which in the multi – user version of EUMEL must be entered in the privileged branch of the task tree. Only then it is guaranteed that the current state of your system is secured.

Otherwise the system is looked upon as being terminated which can be seen from the fact that at the next start the system is back with 'RERUN'. Then you can resume your work only at the last fixpoint and data collected last (usually about 15 minutes) can be lost.

Installation of the EUMEL system

A: Necessary diskettes

- EUMEL system diskette : "EUMELgenerator XY"[1)]
- EUMEL background diskettes : "BG0" ... "BGn"
- EUMEL archive diskettes : "std..."
- EUMEL archive diskette : "XY" (type – dependent)
- MS – DOS diskette : "EUMELstart"

The "EUMELgenerator XY" diskette contains a small, however, complete EUMEL system. After booting this system, the user is able to generate one or more EUMEL partitions. Then these areas are checked in respect to defective tracks and the SHard is installed on the respective partition. **During the generation the diskette must not be write – protected!**

The background diskettes "BG0" ... "BGn" contain the actual operating system EUMEL. These are the system kernel EUMEL 0 and the system parts based upon it (cf. page A – 2).

The standard archive diskettes "std..." contain ELAN program packets and font tables which, after the installation of the operating system, are required for example for a printer installation or extended computing functions. For further detailed information please refer to your User and System Handbook.

The archive diskette "XY" contains ELAN program packets, which supply functions which do not belong to the standard performance of EUMEL or the present SHard version.

1) XY stands for the type designation of a computer like: XT, AT, M24 etc.
The number of diskettes delivered also depends on the type of computer because, for example, 'EUMELstart' is only required if a partitioning is actually possible.

The "EUMELstart" diskette is an MS-DOS diskette and contains command files. The command files enable the user, while working with MS-DOS, to activate one chosen EUMEL partition. At the same time MS-DOS is deactivated and the EUMEL system will be booted after a while.

B: Partitioning the hard disk / SHard installation

If you have already installed an operating system on your hard disk, you have to make sure that there is sufficient space left for a EUMEL system. As, for example, MS-DOS occupies the entire hard disk this system must be secured first, then erased by the MS-DOS command 'fdisk' and installed in an appropriately smaller way. When installing EUMEL you can also erase all systems existing so far; for this the 'EUMELgenerator' offers you the option *Erasing the entire partition table.* In doing so, all existing data is lost. Please make sure that you have secured all data beforehand!

Note: The installation of the SHard can cause problems with hard disks of a capacity of more than 32 megabyte (error message: *disk cannot be read* when searching for defective tracks). Therefore set up your EUMEL partition(s) on the first 32 megabyte.

In order to carry out the partitioning for your EUMEL system, feed the "EUMELgenerator" diskette into the boot disk drive. If the diskette is provided with a write protect mark, this has to be removed beforehand.

Now switch on the computer or press the keyboard RESET by actuating the CTRL ALT DEL keys simultaneously, if your machine has already been switched on.

The 'EUMELgenerator' first answers with the following SHard message:

```
Setup - SHard for EUMEL on XY and compatible V x.x
Copyright (C) 1985,86
EUMEL is loaded from background
```

Afterwards EUMEL 0 messages appear concerning BG, RAM and buffer capacity in relation to the diskette background of the 'EUMELgenerator'.

NOTE: The 'EUMELgenerator' is not to be interrupted during the memory test (stars). If, however, it happens erroneously, then you continue by pressing the 1 key for system start in the start dialog. Then the installation can be continued as usual. Under no circumstances select 2 'load new background from archive', as long as the 'EUMELgenerator' diskette is in the archive drive.

After booting the 'EUMELgenerator' a table is displayed, which tells you whether partitions have already been set up on the hard disk and how they are specified.

Apart from size, start and end track of the individual partitions there is also a type number displayed; the type numbers 69 to 72 are assigned to EUMEL partitions in ascending order, the number 1 or 4 is assigned to MS-DOS depending on the size of the partition already set up. You should remember the type numbers of the partitions set up because these specifications are of importance later on when the whole system is prepared for partition changes. If you set up several EUMEL partitions, then you can only identify them by the type number!

In addition, the partition which is active at the moment is marked by a corrresponding entry in the table. **A partition is called "active" if it is booted after the computer is switched on again or after the next keyboard RESET.**

Now you can select one of the following functions:

- generating a EUMEL partition;
- activating a partition;
- erasing a EUMEL partition;
- erasing the entire partition table;
- ending the generation.

When generating a EUMEL partition only specifications concerning size and start cylinder are requested. For this there are defaults which you can confirm by pressing the CR key.

When resetting, the default for the partition size orientates itself by the largest coherent free space on your disk, the default for the start cylinder orientates itself by the smallest coherent free space on which a partition of the selected size can be set up.

NOTE: If a EUMEL version is to be installed which can only manage 16 megabyte (1.7.3 or 1.8.0), then the partition cannot be set up any larger. Here no general statement can be made on the number of the tracks to be reserved because very different disk partitions are on the market. Please refer to the hard disk manual of your hardware manufacturer for the respective specifications.

Erasing a EUMEL partition is only carried out logically not physically which means that only the entry in the partition table is erased. If you are to set up a new partition at the same position at a later stage and if you have not yet erased this area physically, then after booting the computer, the old system would be restarted. The message 'no EUMEL system found' (cf. below) does not appear then.

After having set up your EUMEL partition(s), make sure that you leave your EUMEL-generator correctly, because, as mentioned previously, it is a complete EUMEL system with fixpoint/rerun logic. 'shutup' is carried out automatically if you select the function '0. end generation'.

If the 'END' message appears on your screen, then this step of the installation is finished. Now you have set up one (or several) EUMEL partitions and installed the SHard. Please remove the 'EUMELgenerator' diskette from the disk drive now.

Loading a EUMEL background

With the next step the EUMEL system is installed on your hard disk, i.e. a background is generated on the hard disk.

After finishing the 'EUMELgenerator' correctly and removing the diskette from the disk drive you have to press the keyboard RESET. This can either be done by pressing the CNTL ALT DEL keys simultaneously on the keyboard or by switching the computer off and on again (please wait a moment between switching off and switching on).

After a short period of time the system answers with the following SHard message:

```
SHard for EUMEL on XY, V x.x
Copyright (C) 1985,86
no EUMEL system found
```

Now put the background diskette 'BG0' into the boot drive and press a key.

The system kernel is now being loaded and the above mentioned specifications concerning BG, RAM and buffer capacity as well as the channels connected appear, this time, however, referring to the set up hard disk partition. Please press a key again during the memory test, in order to get into the start dialog and to avoid that EUMEL 0 tries to start the system. If you missed that, then the message 'invalid background' appears. Then you have again the possibility to get into the start dialog by pressing any key.

Here you select the menu item 2 'load new background from archive' and confirm the question 'overwrite old background' with y for 'yes'.

Now a counter appears on the screen which indicates the read blocks. If your background is distributed over several diskettes, then, upon the inquiry 'new BG archive fed in', you have to feed in the next diskette and answer with y. Please make sure of the numbering of the BG diskettes!

With defective diskettes, read errors may occur; then the system issues the message 'hard read error' or 'softerror'. In the latter case, after several futile attempts, the sector in question could be read after all. In case of a hard read error, the diskette cannot be used.

If all diskettes have been read in, you have to press the keyboard RESET for the last time in order to boot the system. Do not forget to remove the background diskette from the disk drive before doing so.

If you do not press any key during memory test, then the booting is continued and the EUMEL system issues 'system installed'. This takes a few seconds with the background delivered, however, with larger system backups it can also take a couple of minutes; don't lose patience too quickly!

As the delivered background is not configured, the system immediately gets into the 'configurator' when booting for the first time after installation. Now you have to configurate channel 1 as "PC.ascii" or anything appropriate for your individual device. In case you are using a EUMEL version 1.7.3 and therefore do not have the configuration space at your dispense, then configure channel 1 as 'PC' and terminal. For further information, please refer to the System Handbook (Part 1).

An outline of the various steps of installation

1. Feed in the 'EUMELgenerator' diskette into the disk drive.
2. Switch on computer or carry out keyboard RESET with CTRL ALT DEL .
3. Set up EUMEL partition.
4. Finish generation and wait for 'END' message.
5. Remove the 'EUMELgenerator' diskette.
6. Keyboard RESET
7. Wait for 'no EUMEL system found'. When the message 'EUMEL is loaded from background' appears, then continue with 9.
8. Feed in the first background diskette ('BG0') and press a key.
9. Press a key during memory test, in order to get into the start dialog.
10. Select menu item 2 : load new background from archive.
11. If necessary, feed in further BG diskettes upon corresponding request and confirm with y .
12. Keyboard RESET upon corresponding request
13. After booting the system, carry out configuration according to the System Handbook.
14. If necessary, insert ELAN packets for partition changes in the task 'SYSUR'.

If a EUMEL version 1.7.3 is being used,

- first cancel the command *free global manager* in the task 'configurator',
- feed in the archive diskette "XY" and reserve the archive:
  archive ("XY"),
- fetch the file "XY install" from the archive diskette:
  fetch ("XY install", archive),
- start insertion:
  run.

Installing printer software

In order to operate a printer with your EUMEL system, apart from the connection of the printer with a fitting cable you must make the corresponding software available for this printer. The printer adaptations serve this purpose.

The standard archive "std.printer" contains printer adaptations for the activation of various and conventional printer types. If one of those printers is to be connected to the EUMEL system, first of all a task "PRINTER" has to be created as a son task of "SYSUR". This can be done with the supervisor command:

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

Then the following steps have to be carried out in this task:

- reserve archive:
  ```
  archive ("std.printer")
  ```
- fetch printer adaptation from archive:
  ```
  fetch ("printer.printer type", archive)
  ```
- switch off line numbering during insertion:
  ```
  check off
  ```
- insert printer adaptation:
  ```
  insert ("printer.printer type")
  ```

Example:

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

After the insertion the next step consists of the inquiry concerning the printer channel. It should be configurated with the device table 'transparent'. Then, if necesary, printer specific inquiries concerning paper width, positioning procedure or the like, which have to be answered with 'y' or 'n', are made. All alternative answers to the respective inquiry are offered in succession until an alternative is answered with 'y'.

The last step consists of the request to feed in the archive with the required font table. This font table, a description of all displayable characters in all printable font types, is usually on the same diskette as the printer adaptation.

When the generation is finished, the font table has to be set in the multi-user version in all existing tasks – particularly in the task 'PUBLIC'; this can be done with the font table command:

Example:

```
fonttable("fonttab.epson.fx")
```

Then from each task a file can be printed with the command

```
print ("file name")
```

The setting of the font table is above all essential to 'lineform', 'pageform' etc.

If no suitable printer adaptation for the printer to be connected can be found on the standard archive "std.printer", then the printer adaptation "printer.std." should be used. This printer adaptation is a universal one for all printers which perform a 'Carriage Return' (i.e. moving the printhead to the left margin) with ASCII code 13 and which perform a carriage return of 1/6-in. with ASCII code 10. Then with the help of it you can print in a type font (either 10 or 12 characters per inch, depending on the set font table). This way you get at least a minimum control of the printer.

Installing printer software in a single – user version

The installation of the printer software in a single – user version is carried out in a similar way as in the multi – user version. Here only those steps have to be carried out which have to be taken in the task "PRINTER" in the multi – user version. A task "PRINTER" need not be created.