

Quellcode

der insertierten ELAN – Pakete

Version: 1.8.0
Teil: Single – User
Stand: 10.11.86

© 1986
Selbstverlag GMD

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

Herausgeber:

Gesellschaft für Mathematik und Datenverarbeitung mbH

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

Texterstellung:

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

Umschlaggestaltung:

Hannelotte Wecken

Hinweis:

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

Inhaltsverzeichnis

1. Übersicht über die insertierten Pakete
2. Übersicht über die exportierten Objekte nach Paketen geordnet
3. Übersicht über die exportierten Objekte alphabetisch geordnet
4. Quellcode der insertierten Pakete

1. Übersicht über die insertierten Pakete

- '(M)' vor der Paketnummer heißt, daß dies Objekt nur im Multi – User vorhanden ist.
- '(S)' vor der Paketnummer heißt, daß dies Objekt nur im Single – User vorhanden ist.
- '(T)' vor der Paketnummer heißt, daß dies Objekt nur in einem System mit Textverarbeitung vorhanden ist.

Die Paketnummer ergibt sich aus der Reihenfolge, in der die Pakete im Multi – User mit Textverarbeitung insertiert wurden. Bitte beachten Sie, daß diese Reihenfolge nicht der Insertierungsreihenfolge im Single – User entspricht. Der Quellcode der insertierten Pakete ist in Teil 4 nach Paketnummern sortiert.

1. a
2. bits
3. text
4. pcb and init control
5. dataspace
6. basic transput
7. bool
8. integer
9. error handling
10. real
11. date handling
12. command dialogue
13. thesaurus handling
14. local manager
15. pattern match
16. file handling
17. elan do interface
18. scanner
19. screen description
20. tasten verwaltung
21. editor paket
22. editor functions
23. std transput
24. local manager part 2
25. eumel coder part 1
26. mathlib
27. command handler
- S29. advertising
- S30. tasks single
- S31. font store
48. basic archive
- S49. archive single
39. nameset

- S40. system info
- 52. konfigurieren
- S53. configurator single
- S43. single user monitor
- 44. sysgen off
- S45. ur start/S

2. Übersicht über die exportierten Objekte nach Paketen geordnet:

- '(M)' vor der Paketnummer heißt, daß dies Objekt nur im Multi – User vorhanden ist.
- '(S)' vor der Paketnummer heißt, daß dies Objekt nur im Single – User vorhanden ist.
- '(T)' vor der Paketnummer heißt, daß dies Objekt nur in einem System mit Textverarbeitung vorhanden ist.

Die Paketnummer ergibt sich aus der Reihenfolge, in der die Pakete im Multi – User mit Textverarbeitung insertiert wurden. Bitte beachten Sie, daß diese Reihenfolge nicht der Insertierungsreihenfolge im Single – User entspricht. Der Quellcode der insertierten Pakete ist in Teil 4 nach Paketnummern sortiert.

PACKET a :

PACKET bits :

PROC rotate (INT VAR bits, INT CONST number of bits)	2-19
INT OP AND (INT CONST left, right)	2-23
INT OP OR (INT CONST left, right)	2-27
INT OP XOR (INT CONST left, right)	2-31
BOOL PROC bit (INT CONST bits, bit no)	2-35
PROC set bit (INT VAR bits, INT CONST bit no)	2-41
PROC reset bit (INT VAR bits, INT CONST bit no)	2-47
INT PROC lowest set (INT CONST bits)	2-53
INT PROC lowest reset (INT CONST bits)	2-65

PACKET text :

INT CONST max text length	3-35
TEXT OP SUB (TEXT CONST text, INT CONST pos)	3-37
TEXT PROC subtext (TEXT CONST source, INT CONST from, to)	3-41
TEXT PROC subtext (TEXT CONST source, INT CONST from)	3-45
INT PROC code (TEXT CONST text)	3-49
TEXT PROC code (INT CONST code)	3-53
INT OP ISUB (TEXT CONST text, INT CONST index)	3-57
PROC replace (TEXT VAR text, INT CONST index, value)	3-61
REAL OP RSUB (TEXT CONST text, INT CONST index)	3-65
PROC replace (TEXT VAR text, INT CONST index, REAL CONST code)	3-69
PROC replace (TEXT VAR dest, INT CONST pos, TEXT CONST source)	3-74
TEXT PROC text (TEXT CONST source, INT CONST length)	3-78
TEXT PROC text (TEXT CONST source, INT CONST length, from)	3-95
OP CAT (TEXT VAR right, TEXT CONST left)	3-99
TEXT OP + (TEXT CONST left, right)	3-103
TEXT OP * (INT CONST times, TEXT CONST source)	3-109
INT PROC length (TEXT CONST text)	3-120
INT OP LENGTH (TEXT CONST text)	3-124
INT PROC pos (TEXT CONST source, pattern)	3-128
INT PROC pos (TEXT CONST source, pattern, INT CONST from)	3-132
INT PROC pos (TEXT CONST source, pattern, INT CONST from, to)	3-136
INT PROC pos (TEXT CONST source, low, high, INT CONST from)	3-140

TEXT PROC compress (TEXT CONST text)	3-144
PROC change (TEXT VAR destination, INT CONST from, to, TEXT CONST new)	3-167
PROC change (TEXT VAR destination, TEXT CONST old, new)	3-183
PROC delete char (TEXT VAR string, INT CONST delete pos)	3-214
PROC insert char (TEXT VAR string, TEXT CONST char, INT CONST insert pos)	3-224
INT PROC heap size	3-236
PROC collect heap garbage	3-240
PROC stranalyze (ROW 256 INT CONST table, INT VAR sum, INT CONST max sum, TEXT CONST string, INT VAR index, INT CONST to, INT VAR exit code)	3-244
BOOL OP LEXEQUAL (TEXT CONST left, right)	3-262
BOOL OP LEXGREATER (TEXT CONST left, right)	3-269
BOOL OP LEXGREATEREQUAL (TEXT CONST left, right)	3-276
OP CAT (TEXT VAR result, INT CONST number)	3-374
PROC insert int (TEXT VAR result, INT CONST insert pos, number)	3-379
PROC delete int (TEXT VAR result, INT CONST delete pos)	3-385

PACKET pcb and init control :

TYPE INITFLAG	4-19
INT PROC session	4-22
INT PROC pcb (INT CONST field)	4-26
PROC set line nr (INT CONST value)	4-34
OP := (INITFLAG VAR flag, BOOL CONST flagtrue)	4-39
BOOL PROC initialized (INITFLAG VAR flag)	4-50
REAL PROC clock (INT CONST nr)	4-62
PROC storage (INT VAR size, used)	4-66
INT PROC id (INT CONST no)	4-70
PROC ke	4-74

PACKET dataspace :

TYPE ALIGN	5-21
OP := (DATASPACE VAR dest, DATASPACE CONST source)	5-23
DATASPACE PROC nilspace	5-27
PROC forget (DATASPACE CONST dataspace)	5-31
PROC type (DATASPACE CONST ds, INT CONST type)	5-35
INT PROC type (DATASPACE CONST ds)	5-39
INT PROC heap size (DATASPACE CONST ds)	5-43
INT PROC storage (DATASPACE CONST ds)	5-47
INT PROC ds pages (DATASPACE CONST ds)	5-51
INT PROC next ds page (DATASPACE CONST ds, INT CONST page nr)	5-59
PROC blackout (DATASPACE CONST ds, INT CONST page nr, code1, code2, INT VAR return code)	5-63
PROC blockin (DATASPACE VAR ds, INT CONST page nr, code1, code2, INT VAR return code)	5-68

PACKET basic transput :

PROC out (TEXT CONST text)	6-35
PROC outsubtext (TEXT CONST source, INT CONST from)	6-39
PROC outsubtext (TEXT CONST source, INT CONST from, to)	6-43
PROC outtext (TEXT CONST source, INT CONST from, to)	6-47
OP TIMESOUT (INT CONST times, TEXT CONST text)	6-59
PROC display (TEXT CONST text)	6-81
PROC inchar (TEXT VAR character)	6-87
TEXT PROC incharety	6-91

exportierte Objekte nach Paketen geordnet

TEXT PROC incharety (INT CONST time limit)	6-95
PROC pause (INT CONST time limit)	6-100
PROC pause	6-105
PROC cat input (TEXT VAR t, esc char)	6-113
PROC cursor (INT CONST x, y)	6-118
PROC get cursor (INT VAR x, y)	6-124
PROC cout (INT CONST number)	6-128
INT PROC channel	6-133
BOOL PROC online	6-137
PROC control (INT CONST code1, code2, code3, INT VAR return code)	6-142
PROC blackout (ROW 256 INT CONST block, INT CONST code1, code2, INT VAR return code)	6-146
PROC blockin (ROW 256 INT VAR block, INT CONST code1, code2, INT VAR return code)	6-161

PACKET bool :

BOOL CONST true	7-4
BOOL CONST false	7-4
BOOL OP XOR (BOOL CONST left, right)	7-7

PACKET integer :

INT PROC minint	8-7
INT PROC maxint	8-9
TEXT PROC text (INT CONST number)	8-12
TEXT PROC text (INT CONST number, length)	8-25
INT PROC int (TEXT CONST number)	8-38
INT OP MOD (INT CONST left, right)	8-95
INT PROC sign (INT CONST argument)	8-101
INT OP SIGN (INT CONST argument)	8-110
INT PROC abs (INT CONST argument)	8-114
INT OP ABS (INT CONST argument)	8-122
INT OP ** (INT CONST arg, exp)	8-126
INT PROC min (INT CONST first, second)	8-154
INT PROC max (INT CONST first, second)	8-160
BOOL PROC last conversion ok	8-170
PROC set conversion (BOOL CONST success)	8-174
PROC initialize random (INT CONST start)	8-197
INT PROC random (INT CONST lower bound, upper bound)	8-208

PACKET error handling :

PROC enable stop	9-28
PROC disable stop	9-32
BOOL PROC is error	9-40
PROC clear error	9-44
TEXT PROC error message	9-72
INT PROC error code	9-79
INT PROC error line	9-85
PROC errorstop (TEXT CONST message)	9-101
PROC errorstop (INT CONST code, TEXT CONST message)	9-107
PROC put error	9-116
PROC stop	9-135

PACKET real :

REAL PROC max real	10-40
REAL PROC small real	10-42
INT PROC decimal exponent (REAL CONST mantissa)	10-48
PROC set exp (INT CONST exponent, REAL VAR number)	10-52
REAL PROC floor (REAL CONST real)	10-62
REAL PROC round (REAL CONST real, INT CONST digits)	10-66
TEXT PROC text (REAL CONST real)	10-88
TEXT PROC text (REAL CONST real, INT CONST length)	10-155
TEXT PROC text (REAL CONST real, INT CONST length, frags)	10-199
REAL PROC real (TEXT CONST text)	10-243
REAL PROC abs (REAL CONST value)	10-329
REAL OP ABS (REAL CONST value)	10-338
INT PROC sign (REAL CONST value)	10-344
INT OP SIGN (REAL CONST value)	10-353
REAL OP MOD (REAL CONST left, right)	10-359
REAL PROC frac (REAL CONST value)	10-369
REAL PROC max (REAL CONST a, b)	10-375
REAL PROC min (REAL CONST a, b)	10-381
OP INCR (REAL VAR dest, REAL CONST increment)	10-387
OP DECR (REAL VAR dest, REAL CONST decrement)	10-393
INT PROC int (REAL CONST value)	10-399
REAL PROC real (INT CONST value)	10-424

PACKET date handling :

REAL PROC day	11-28
REAL PROC hour	11-29
REAL PROC minute	11-30
REAL PROC second	11-31
TEXT PROC date	11-33
TEXT PROC date (REAL CONST datum)	11-44
TEXT PROC day (REAL CONST datum)	11-128
TEXT PROC month (REAL CONST datum)	11-139
TEXT PROC year (REAL CONST datum)	11-156
TEXT PROC time of day	11-166
TEXT PROC time of day (REAL CONST value)	11-170
TEXT PROC time (REAL CONST value)	11-174
TEXT PROC time (REAL CONST value, INT CONST length)	11-178
REAL PROC date (TEXT CONST datum)	11-201
REAL PROC time (TEXT CONST time)	11-260
REAL CONST hour	11-265

PACKET command dialogue :

TYPE QUIET	12-29
QUIET PROC quiet	12-31
BOOL PROC command dialogue	12-36
PROC command dialogue (BOOL CONST status)	12-40
BOOL PROC yes (TEXT CONST question)	12-45
BOOL PROC no (TEXT CONST question)	12-81
PROC say (TEXT CONST message)	12-87
PROC param position (INT CONST x)	12-95
TEXT PROC last param	12-101
PROC last param (TEXT CONST new)	12-114
TEXT PROC std	12-118

exportierte Objekte nach Paketen geordnet

PACKET thesaurus handling :

TYPE THESAURUS	13-17
THESAURUS PROC empty thesaurus	13-96
OP := (THESAURUS VAR dest, THESAURUS CONST source)	13-103
PROC insert (THESAURUS VAR thesaurus, TEXT CONST name, INT VAR index)	13-111
PROC insert (THESAURUS VAR thesaurus, TEXT CONST name)	13-170
PROC delete (THESAURUS VAR thesaurus, TEXT CONST name, INT VAR index)	13-180
PROC delete (THESAURUS VAR thesaurus, INT CONST index)	13-188
BOOL OP CONTAINS (THESAURUS CONST thesaurus, TEXT CONST name)	13-231
PROC rename (THESAURUS VAR thesaurus, TEXT CONST old, new)	13-242
PROC rename (THESAURUS VAR thesaurus, INT CONST index, TEXT CONST new)	13-248
INT PROC link (THESAURUS CONST thesaurus, TEXT CONST name)	13-267
TEXT PROC name (THESAURUS CONST thesaurus, INT CONST index)	13-274
PROC get (THESAURUS CONST thesaurus, TEXT VAR name, INT VAR index)	13-281
INT PROC highest entry (THESAURUS CONST thesaurus)	13-323

PACKET local manager :

PROC create (TEXT CONST name)	14-61
DATASPACE PROC new (TEXT CONST name)	14-86
DATASPACE PROC old (TEXT CONST name)	14-96
DATASPACE PROC old (TEXT CONST name, INT CONST expected type)	14-110
BOOL PROC exists (TEXT CONST name)	14-127
PROC forget (TEXT CONST name)	14-134
PROC forget (TEXT CONST name, QUIET CONST q)	14-145
PROC forget	14-157
PROC status (TEXT CONST name, status text)	14-166
TEXT PROC status (TEXT CONST name)	14-176
PROC status (INT CONST pos, TEXT CONST status pattern)	14-187
PROC copy (DATASPACE CONST source, TEXT CONST dest name)	14-200
PROC copy (TEXT CONST source name, dest name)	14-218
PROC rename (TEXT CONST old name, new name)	14-224
PROC begin list	14-237
PROC get list entry (TEXT VAR entry, status text)	14-244
TEXT PROC write password	14-257
TEXT PROC read password	14-263
PROC enter password (TEXT CONST password)	14-270
PROC enter password (TEXT CONST file name, write pass, read pass)	14-284
BOOL PROC read permission (TEXT CONST name, supply password)	14-312
BOOL PROC write permission (TEXT CONST name, supply password)	14-335
THESAURUS PROC all	14-358

PACKET pattern match :

TEXT OP - (TEXT CONST alphabet)	15-43
TEXT OP OR (TEXT CONST a, b)	15-54
TEXT OP ** (TEXT CONST p, INT CONST x)	15-58
TEXT CONST any	15-62
TEXT PROC any (INT CONST n)	15-64
TEXT PROC any (TEXT CONST a)	15-70
TEXT PROC any (INT CONST n, TEXT CONST a)	15-72
TEXT PROC notion (TEXT CONST t)	15-78
TEXT CONST bound	15-87
TEXT PROC match (INT CONST x)	15-91
INT PROC matchpos (INT CONST x)	15-95
INT PROC matchend (INT CONST x)	15-97

exportierte Objekte nach Paketen geordnet

TEXT PROC somefix (TEXT CONST pattern)	15-123
BOOL OP UNLIKE (TEXT CONST t, p)	15-191
BOOL OP LIKE (TEXT CONST t, pattern)	15-193
TEXT PROC notion (TEXT CONST t, INT CONST r)	15-765

PACKET file handling :

TYPE FILE	16-150
TYPE FRANGE	16-152
OP := (FILE VAR left, FILE CONST right)	16-160
TEXT PROC prefix (TEXT CONST pattern)	16-546
BOOL PROC pattern found	16-674
TRANSPUTDIRECTION PROC input	16-851
TRANSPUTDIRECTION PROC output	16-856
TRANSPUTDIRECTION PROC modify	16-861
FILE PROC sequential file (TRANSPUTDIRECTION CONST mode, DATASPACE CONST ds)	16-869
FILE PROC sequential file (TRANSPUTDIRECTION CONST mode, TEXT CONST name)	16-885
PROC reset (FILE VAR f)	16-926
PROC reset (FILE VAR f, TRANSPUTDIRECTION CONST mode)	16-935
PROC input (FILE VAR f)	16-962
PROC output (FILE VAR f)	16-969
PROC modify (FILE VAR f)	16-976
PROC close (FILE VAR f)	16-983
PROC to line (FILE VAR f, INT CONST destination line)	16-1023
PROC to first record (FILE VAR f)	16-1031
PROC to eof (FILE VAR f)	16-1038
PROC putline (FILE VAR f, TEXT CONST word)	16-1045
PROC delete record (FILE VAR f)	16-1055
PROC insert record (FILE VAR f)	16-1063
PROC down (FILE VAR f)	16-1071
PROC up (FILE VAR f)	16-1078
PROC down (FILE VAR f, INT CONST n)	16-1085
PROC up (FILE VAR f, INT CONST n)	16-1091
PROC write record (FILE VAR f, TEXT CONST record)	16-1098
PROC read record (FILE CONST f, TEXT VAR record)	16-1111
PROC line (FILE VAR f)	16-1119
PROC line (FILE VAR f, INT CONST lines)	16-1145
PROC getline (FILE VAR f, TEXT VAR text)	16-1152
BOOL PROC is first record (FILE CONST f)	16-1174
BOOL PROC eof (FILE CONST f)	16-1182
INT PROC line no (FILE CONST f)	16-1197
PROC line type (FILE VAR f, INT CONST t)	16-1204
INT PROC line type (FILE CONST f)	16-1210
PROC put (FILE VAR f, TEXT CONST word)	16-1217
PROC put (FILE VAR f, INT CONST value)	16-1236
PROC put (FILE VAR f, REAL CONST real)	16-1243
PROC write (FILE VAR f, TEXT CONST word)	16-1250
PROC get (FILE VAR f, TEXT VAR word, TEXT CONST separator)	16-1268
PROC get (FILE VAR f, TEXT VAR word, INT CONST max length)	16-1312
PROC get (FILE VAR f, TEXT VAR word)	16-1336
PROC get (FILE VAR f, INT VAR number)	16-1346
PROC get (FILE VAR f, REAL VAR number)	16-1354
PROC split line (FILE VAR f, INT CONST split col)	16-1365
PROC split line (FILE VAR f, INT CONST split col, BOOL CONST note indentation)	16-1371
PROC concatenate line (FILE VAR f, BOOL CONST delete blanks)	16-1407
PROC set range (FILE VAR f, INT CONST start line, start col, FRANGE VAR old range)	16-1550

exportierte Objekte nach Paketen geordnet

PROC set range (FILE VAR f, FRANGE VAR new range)	16-1620
PROC reset range (FILE VAR f)	16-1668
PROC remove (FILE VAR f, INT CONST size)	16-1679
PROC clear removed (FILE VAR f)	16-1687
PROC reinsert (FILE VAR f)	16-1695
PROC copy attributes (FILE CONST source file, FILE VAR dest file)	16-1703
INT PROC max line length (FILE CONST f)	16-1716
PROC max line length (FILE VAR f, INT CONST new limit)	16-1723
TEXT PROC headline (FILE CONST f)	16-1732
PROC headline (FILE VAR f, TEXT CONST head)	16-1739
PROC get tabs (FILE CONST f, TEXT VAR tabs)	16-1746
PROC put tabs (FILE VAR f, TEXT CONST tabs)	16-1753
INT PROC edit info (FILE CONST f)	16-1760
PROC edit info (FILE VAR f, INT CONST info)	16-1767
INT PROC lines (FILE CONST f)	16-1774
INT PROC removed lines (FILE CONST f)	16-1781
INT PROC segments (FILE CONST f)	16-1788
INT PROC col (FILE CONST f)	16-1795
PROC col (FILE VAR f, INT CONST new column)	16-1801
TEXT PROC word (FILE CONST f)	16-1809
TEXT PROC word (FILE CONST f, TEXT CONST delimiter)	16-1815
TEXT PROC word (FILE CONST f, INT CONST max length)	16-1825
BOOL PROC at (FILE CONST f, TEXT CONST word)	16-1831
PROC exec (PROC (TEXT VAR, TEXT CONST) proc, FILE VAR f, TEXT CONST t)	16-1844
PROC exec (PROC (TEXT VAR, INT CONST) proc, FILE VAR f, INT CONST i)	16-1853
INT PROC pos (FILE CONST f, TEXT CONST pattern, INT CONST i)	16-1861
PROC down (FILE VAR f, TEXT CONST pattern)	16-1869
PROC down (FILE VAR f, TEXT CONST pattern, INT CONST max line)	16-1875
PROC downety (FILE VAR f, TEXT CONST pattern)	16-1884
PROC downety (FILE VAR f, TEXT CONST pattern, INT CONST max line)	16-1890
PROC up (FILE VAR f, TEXT CONST pattern)	16-1899
PROC up (FILE VAR f, TEXT CONST pattern, INT CONST max line)	16-1905
PROC uppety (FILE VAR f, TEXT CONST pattern)	16-1914
PROC uppety (FILE VAR f, TEXT CONST pattern, INT CONST max line)	16-1920
INT PROC len (FILE CONST f)	16-1930
TEXT PROC subtext (FILE CONST f, INT CONST from, to)	16-1938
PROC change (FILE VAR f, INT CONST from, to, TEXT CONST new)	16-1946
BOOL PROC mark (FILE CONST f)	16-1956
PROC mark (FILE VAR f, INT CONST line no, col)	16-1962
INT PROC mark line no (FILE CONST f)	16-1973
INT PROC mark col (FILE CONST f)	16-1982
PROC set marked range (FILE VAR f, FRANGE VAR old range)	16-1993
PROC sort (TEXT CONST dateiname)	16-2016
PROC sort (TEXT CONST dateiname, INT CONST sortieranfang)	16-2020
PROC lex sort (TEXT CONST dateiname)	16-2025
PROC lex sort (TEXT CONST dateiname, INT CONST sortieranfang)	16-2029

PACKET elan do interface :

PROC do (TEXT CONST command)	17-23
PROC no do again	17-44

PACKET scanner :

PROC scan (TEXT CONST scan text)	18-36
PROC continue scan (TEXT CONST scan text)	18-44
PROC next symbol (TEXT VAR symbol)	18-52
PROC next symbol (TEXT VAR symbol, INT VAR type)	18-59

exportierte Objekte nach Paketen geordnet

PROC scan (FILE VAR f)	18-296
PROC next symbol (FILE VAR f, TEXT VAR symbol)	18-303
PROC next symbol (FILE VAR f, TEXT VAR symbol, INT VAR type)	18-312

PACKET screen description :

INT PROC xsize	19-9
INT PROC ysize	19-11
INT PROC marksize	19-13
PROC xsize (INT CONST i)	19-15
PROC ysize (INT CONST i)	19-17
PROC marksize (INT CONST i)	19-19
BOOL PROC mark refresh line mode	19-24
PROC mark refresh line mode (BOOL CONST b)	19-28

PACKET tasten verwaltung :

PROC lernsequenz auf taste legen (TEXT CONST taste, lernsequenz)	20-27
TEXT PROC lernsequenz auf taste (TEXT CONST taste)	20-63
PROC kommando auf taste legen (TEXT CONST taste, kommando)	20-71
TEXT PROC kommando auf taste (TEXT CONST taste)	20-79
BOOL PROC taste enthaelt kommando (TEXT CONST taste)	20-87
PROC std tastenbelegung	20-92

PACKET editor paket :

PROC editget command (BOOL CONST schalter)	21-81
PROC editget (TEXT VAR editsatz, INT CONST editlimit, editlaenge, TEXT CONST sep, res, TEXT VAR exit char)	21-87
PROC editget (TEXT VAR editsatz, INT CONST editlimit, TEXT VAR exit char)	21-192
PROC editget (TEXT VAR editsatz, TEXT CONST sep, res, TEXT VAR exit char)	21-196
PROC editget (TEXT VAR editsatz)	21-200
PROC editget (TEXT VAR editsatz, INT CONST editlimit, editlaenge)	21-205
BOOL PROC is editget	21-913
PROC get editline (TEXT VAR editline, INT VAR editpos, editmarke)	21-917
PROC put editline (TEXT CONST editline, INT CONST editpos, editmarke)	21-925
BOOL PROC within kanji (TEXT CONST satz, INT CONST stelle)	21-933
BOOL PROC is kanji esc (TEXT CONST char)	21-947
BOOL PROC two bytes	21-952
PROC two bytes (BOOL CONST new mode)	21-954
BOOL PROC write permission	21-989
PROC push (TEXT CONST ausfuehrkommando)	21-991
PROC type (TEXT CONST ausfuehrkommando)	21-1009
PROC getchar (TEXT VAR zeichen)	21-1173
BOOL PROC is incharety (TEXT CONST muster)	21-1194
TEXT PROC getcharety	21-1208
PROC get editcursor (INT VAR x, y)	21-1219
INT PROC aktueller editor	21-1279
INT PROC groesster editor	21-1281
PROC quit last	21-2144
PROC quit	21-2153
INT CONST aktueller editor	21-2158
PROC nichts neu	21-2217
PROC satznr neu	21-2219
PROC ueberschrift neu	21-2221
PROC zeile neu	21-2223
PROC abschnitt neu (INT CONST von satznr, bis satznr)	21-2228

exportierte Objekte nach Paketen geordnet

PROC bildabschnitt neu (INT CONST von zeile, bis zeile)	21-2237
PROC bild neu	21-2249
PROC bild neu (FILE VAR f)	21-2251
PROC alles neu	21-2261
PROC satznr zeigen	21-2270
PROC ueberschrift zeigen	21-2274
PROC bild zeigen	21-2417
PROC set busy indicator	21-2508
PROC word wrap (BOOL CONST b)	21-2607
BOOL PROC word wrap	21-2623
INT PROC margin	21-2630
PROC margin (INT CONST i)	21-2632
BOOL PROC rubin mode	21-2653
BOOL PROC rubin mode (INT CONST editor nr)	21-2655
PROC edit (INT CONST i, TEXT CONST res, PROC (TEXT CONST) kommando interpreter)	21-2665
PROC edit (INT CONST von, bis, start, TEXT CONST res, PROC (TEXT CONST) kommando interpreter)	21-2670
PROC open editor (FILE CONST new file, BOOL CONST access)	21-2753
PROC open editor (INT CONST editor nr, FILE CONST new file, BOOL CONST access, INT CONST x start, y, x len start, y len)	21-2792
PROC open editor (INT CONST i)	21-2881
FILE PROC editfile	21-2929
PROC get window (INT VAR x, y, x size, y size)	21-2935

PACKET editor functions :

PROC std kommando interpreter (TEXT CONST taste)	22-58
PROC edit (FILE VAR f)	22-176
PROC edit (FILE VAR f, INT CONST x, y, x size, y size)	22-186
PROC edit (FILE VAR f, TEXT CONST res, PROC (TEXT CONST) kdo interpreter)	22-193
PROC edit	22-200
PROC edit (TEXT CONST filename)	22-233
PROC edit (TEXT CONST filename, INT CONST x, y, x size, y size)	22-250
PROC edit (INT CONST i)	22-261
PROC show (FILE VAR f)	22-266
PROC show (TEXT CONST filename)	22-273
PROC show	22-283
OP PUT (TEXT CONST filename)	22-293
OP P (TEXT CONST filename)	22-362
OP GET (TEXT CONST filename)	22-367
OP G (TEXT CONST filename)	22-438
INT PROC len	22-443
PROC col (INT CONST stelle)	22-448
INT PROC col	22-453
PROC limit (INT CONST limit)	22-458
INT PROC limit	22-463
INT PROC lines	22-468
INT PROC line no	22-473
PROC to line (INT CONST satz nr)	22-478
OP T (INT CONST satz nr)	22-488
PROC down (INT CONST anz)	22-493
OP D (INT CONST anz)	22-498
PROC up (INT CONST anz)	22-503
OP U (INT CONST anz)	22-508
PROC down (TEXT CONST muster)	22-513
OP D (TEXT CONST muster)	22-525
PROC down (TEXT CONST muster, INT CONST anz)	22-530

exportierte Objekte nach Paketen geordnet

PROC up (TEXT CONST muster)	22-535
OP U (TEXT CONST muster)	22-547
PROC up (TEXT CONST muster, INT CONST anz)	22-552
PROC downety (TEXT CONST muster)	22-557
PROC downety (TEXT CONST muster, INT CONST anz)	22-565
PROC uppety (TEXT CONST muster)	22-570
PROC uppety (TEXT CONST muster, INT CONST anz)	22-578
OP C (TEXT CONST old, new)	22-583
OP C (TEXT CONST replacement)	22-587
PROC change to (TEXT CONST old, new)	22-596
OP CA (TEXT CONST old, new)	22-613
PROC change all (TEXT CONST old, new)	22-618
BOOL PROC eof	22-623
BOOL PROC mark	22-628
PROC mark (BOOL CONST mark on)	22-633
BOOL PROC at (TEXT CONST pattern)	22-642
TEXT PROC word	22-646
TEXT PROC word (TEXT CONST sep)	22-651
TEXT PROC word (INT CONST len)	22-656
PROC note (TEXT CONST text)	22-671
PROC note (INT CONST number)	22-677
PROC note line	22-683
BOOL PROC anything noted	22-689
FILE PROC note file	22-695
PROC note edit (FILE VAR context)	22-701
PROC note edit	22-731

PACKET std transput :

PROC sysout (TEXT CONST file name)	23-33
TEXT PROC sysout	23-44
PROC sysin (TEXT CONST file name)	23-48
TEXT PROC sysin	23-59
PROC put (TEXT CONST word)	23-64
PROC put (INT CONST number)	23-73
PROC put (REAL CONST number)	23-79
PROC putline (TEXT CONST textline)	23-85
PROC line	23-94
PROC line (INT CONST times)	23-103
PROC page	23-112
PROC write (TEXT CONST word)	23-120
PROC get (TEXT VAR word)	23-130
PROC get (TEXT VAR word, TEXT CONST separator)	23-152
PROC get (INT VAR number)	23-175
PROC get (REAL VAR number)	23-182
PROC get (TEXT VAR word, INT CONST length)	23-189
PROC getline (TEXT VAR textline)	23-203
PROC get secret line (TEXT VAR textline)	23-217

PACKET local manager part 2 :

PROC list	24-10
PROC list (FILE VAR f)	24-22

PACKET eumel coder part 1 :

PROC help (TEXT CONST proc name)	25-328
PROC bulletin (TEXT CONST packet name)	25-559
PROC bulletin	25-679
PROC packets	25-701
PROC run (TEXT CONST file name)	25-735
PROC run	25-744
PROC run again	25-748
PROC insert (TEXT CONST file name)	25-756
PROC insert	25-765
PROC prot (TEXT CONST file name)	25-826
PROC prot off	25-832
BOOL PROC prot	25-837
PROC check on	25-841
PROC check off	25-845
BOOL PROC check	25-849
PROC warnings on	25-853
PROC warnings off	25-857
BOOL PROC warnings	25-861

PACKET mathlib :

REAL PROC pi	26-25
REAL PROC e	26-26
REAL PROC ln (REAL CONST x)	26-28
REAL PROC log10 (REAL CONST x)	26-32
REAL PROC log2 (REAL CONST z)	26-36
REAL PROC sqrt (REAL CONST z)	26-64
REAL PROC exp (REAL CONST z)	26-83
REAL PROC tan (REAL CONST x)	26-111
REAL PROC tand (REAL CONST x)	26-116
REAL PROC sin (REAL CONST x)	26-146
REAL PROC sind (REAL CONST x)	26-154
REAL PROC cos (REAL CONST x)	26-162
REAL PROC cosd (REAL CONST x)	26-170
REAL PROC arctan (REAL CONST y)	26-204
REAL PROC arctand (REAL CONST x)	26-218
REAL OP ** (REAL CONST b, e)	26-222
REAL OP ** (REAL CONST a, INT CONST b)	26-231
REAL PROC random	26-259
PROC initializerandom (REAL CONST z)	26-263

PACKET command handler :

PROC get command (TEXT CONST command text)	27-33
PROC get command (TEXT CONST command text, TEXT VAR command line)	27-39
PROC analyze command (TEXT CONST command list, INT CONST permitted type, INT VAR command index, number of params, TEXT VAR param 1, param 2)	27-106
PROC analyze command (TEXT CONST command list, command line, INT CONST permitted type, INT VAR command index, number of params, TEXT VAR param 1, param 2)	27-117
PROC do command	27-233
PROC command error	27-247
PROC cover tracks	27-266
PROC cover tracks (TEXT VAR secret)	27-277

exportierte Objekte nach Paketen geordnet

PACKET advertising :

SOME PROC eumel must advertise

S29-4

PACKET tasks single :

TYPE TASK	S30-38
TASK PROC myself	S30-44
OP := (TASK VAR dest, TASK CONST source)	S30-51
BOOL OP = (TASK CONST left, right)	S30-57
BOOL PROC is niltask (TASK CONST t)	S30-63
INT PROC pcb (TASK CONST id, INT CONST field)	S30-69
INT PROC status (TASK CONST id)	S30-75
INT PROC channel (TASK CONST id)	S30-81
REAL PROC clock (TASK CONST id)	S30-87
INT PROC storage (TASK CONST id)	S30-93
PROC continue (INT CONST channel no)	S30-112
INT PROC dataspace	S30-121

PACKET font store :

PROC font table (TEXT CONST new font table)	S31-88
TEXT PROC font table	S31-128
PROC list font tables	S31-135
PROC list fonts (TEXT CONST name)	S31-164
PROC list fonts	S31-176
INT PROC x step conversion (REAL CONST cm)	S31-218
REAL PROC x step conversion (INT CONST steps)	S31-229
INT PROC y step conversion (REAL CONST cm)	S31-237
REAL PROC y step conversion (INT CONST steps)	S31-248
TEXT PROC on string (INT CONST modification)	S31-256
TEXT PROC off string (INT CONST modification)	S31-270
INT PROC font (TEXT CONST font name)	S31-284
TEXT PROC font (INT CONST font number)	S31-298
BOOL PROC font exists (TEXT CONST font name)	S31-311
BOOL PROC next larger font exists (INT CONST font number, INT VAR next larger font)	S31-318
BOOL PROC next smaller font exists (INT CONST font number, INT VAR next smaller font)	S31-338
INT PROC font lead (INT CONST font number)	S31-358
INT PROC font height (INT CONST font number)	S31-371
INT PROC font depth (INT CONST font number)	S31-384
INT PROC indentation pitch (INT CONST font number)	S31-397
INT PROC char pitch (INT CONST font number, TEXT CONST char)	S31-410
INT PROC extended char pitch (INT CONST font number, TEXT CONST esc char, char)	S31-430
TEXT PROC replacement (INT CONST font number, TEXT CONST char)	S31-452
TEXT PROC extended replacement (INT CONST font number, TEXT CONST esc char, char)	S31-480
TEXT PROC font string (INT CONST font number)	S31-537
TEXT PROC y offsets (INT CONST font number)	S31-550
INT PROC bold offset (INT CONST font number)	S31-563
PROC get font (INT CONST font number, INT VAR indentation pitch, font lead, font height, font depth, ROW 256 INT VAR pitch table)	S31-576
PROC get replacements (INT CONST font number, TEXT VAR replacements, ROW 256 INT VAR replacements table)	S31-595

exportierte Objekte nach Paketen geordnet

PACKET basic archive :

INT PROC block number	48-43
PROC seek (INT CONST block)	48-47
PROC rewind	48-51
PROC skip dataspace	48-59
PROC read (DATASPACE VAR ds)	48-70
PROC read (DATASPACE VAR ds, INT CONST max pages, BOOL CONST error accept)	48-74
PROC check read	48-118
PROC write (DATASPACE CONST ds)	48-134
PROC read block (DATASPACE VAR ds, INT CONST ds page no, INT CONST block no, INT VAR return code)	48-268
PROC write block (DATASPACE CONST ds, INT CONST ds page no, INT CONST mode, INT CONST block no, INT VAR return code)	48-292
INT PROC size (INT CONST key)	48-321
INT PROC archive blocks	48-329
PROC search dataspace (INT VAR ds pages)	48-333
PROC format archive (INT CONST format code)	48-378

PACKET archive single :

PROC archive (TEXT CONST name)	S49-72
PROC release (TASK CONST t)	S49-81
PROC fetch (TEXT CONST file name)	S49-215
PROC fetch (TEXT CONST file name, TASK CONST from)	S49-221
PROC erase	S49-308
PROC erase (TEXT CONST file name)	S49-314
PROC erase (TEXT CONST file name, TASK CONST dest)	S49-320
PROC save	S49-411
PROC save (TEXT CONST file name)	S49-417
PROC save (TEXT CONST file name, TASK CONST to)	S49-423
PROC check (TEXT CONST name, TASK CONST from)	S49-534
BOOL PROC exists (TEXT CONST name, TASK CONST from)	S49-563
PROC list (TASK CONST from)	S49-576
PROC list (FILE VAR list file, TASK CONST from)	S49-588
THESAURUS OP ALL (TASK CONST from)	S49-649
PROC clear (TASK CONST dest)	S49-678
PROC format (INT CONST format code, TASK CONST dest)	S49-722
PROC format (TASK CONST dest)	S49-740

PACKET name set :

THESAURUS OP + (THESAURUS CONST left, right)	39-32
THESAURUS OP + (THESAURUS CONST left, TEXT CONST right)	39-47
THESAURUS OP - (THESAURUS CONST left, right)	39-57
THESAURUS OP - (THESAURUS CONST left, TEXT CONST right)	39-72
THESAURUS OP / (THESAURUS CONST left, right)	39-81
THESAURUS OP ALL (TEXT CONST file name)	39-96
THESAURUS OP SOME (THESAURUS CONST thesaurus)	39-105
THESAURUS OP SOME (TASK CONST task)	39-130
THESAURUS OP SOME (TEXT CONST file name)	39-136
THESAURUS OP LIKE (THESAURUS CONST thesaurus, TEXT CONST pattern)	39-142
THESAURUS PROC remainder	39-157
PROC do (PROC (TEXT CONST) operate, THESAURUS CONST thesaurus)	39-163
PROC do (PROC (TEXT CONST, TASK CONST) operate, THESAURUS CONST thesaurus, TASK CONST task)	39-199

exportierte Objekte nach Paketen geordnet

OP FILLBY (THESAURUS VAR thesaurus, FILE VAR file)	39-237
OP FILLBY (FILE VAR file, THESAURUS CONST thesaurus)	39-254
OP FILLBY (TEXT CONST file name, THESAURUS CONST thesaurus)	39-267
PROC fetch (THESAURUS CONST nameset)	39-276
PROC fetch (THESAURUS CONST nameset, TASK CONST task)	39-282
PROC save (THESAURUS CONST nameset)	39-288
PROC save (THESAURUS CONST nameset, TASK CONST task)	39-294
PROC fetch all	39-300
PROC fetch all (TASK CONST manager)	39-306
PROC save all	39-312
PROC save all (TASK CONST manager)	39-318
PROC forget (THESAURUS CONST nameset)	39-324
PROC erase (THESAURUS CONST nameset)	39-330
PROC erase (THESAURUS CONST nameset, TASK CONST task)	39-336
PROC insert (THESAURUS CONST nameset)	39-342
PROC edit (THESAURUS CONST nameset)	39-348

PACKET system info :

PROC task status	S40-34
PROC storage info	S40-45
PROC help	S40-61
PROC help (FILE VAR help file)	S40-71

PACKET konfigurieren :

PROC new configuration	52-39
PROC flow (INT CONST nr, INT CONST dtype)	52-68
PROC ysize (INT CONST channel, new size, INT VAR old size)	52-72
PROC input buffer size (INT CONST nr, size)	52-76
PROC baudrate (INT CONST nr, rate)	52-81
PROC bits (INT CONST channel, number, parity)	52-85
PROC bits (INT CONST channel, key)	52-89
PROC new type (TEXT CONST dtype)	52-105
PROC link (INT CONST nr, TEXT CONST dtype)	52-142
PROC enter outcode (INT CONST eumel code, ziel code)	52-156
PROC enter outcode (INT CONST eumel code, wartezeit, TEXT CONST sequenz)	52-178
PROC enter outcode (INT CONST eumelcode, TEXT CONST wert)	52-194
PROC enter incode (INT CONST elan code, TEXT CONST sequenz)	52-211
PROC cursor logic (INT CONST dist, TEXT CONST pre, mid, post)	52-225
PROC ansi cursor (TEXT CONST pre, mid, post)	52-231
PROC cursor logic (INT CONST dist, modus, TEXT CONST pre, mid, post)	52-237
PROC elbit cursor	52-247

PACKET configurator single :

PROC configurate	S53-220
PROC exec configuration	S53-447
PROC setup	S53-453

PACKET single user monitor :

PROC monitor	S43-34
PROC monitor (PROC init system)	S43-40
PROC set date	S43-121

exportierte Objekte nach Paketen geordnet

PROC shutup	S43-182
PROC save system	S43-192
PROC collect garbage blocks	S43-202
PROC fixpoint	S43-208
PROC set clock (REAL CONST time)	S43-235

exportierte Objekte nach Paketen geordnet

3. Übersicht über die exportierten Objekte alphabetisch geordnet:

- '(M)' vor der Paketnummer heißt, daß dies Objekt nur im Multi – User vorhanden ist.
'(S)' vor der Paketnummer heißt, daß dies Objekt nur im Single – User vorhanden ist.
'(T)' vor der Paketnummer heißt, daß dies Objekt nur in einem System mit Textverarbeitung vorhanden ist.

Die Paketnummer ergibt sich aus der Reihenfolge, in der die Pakete im Multi – User mit Textverarbeitung insertiert wurden. Bitte beachten Sie, daß diese Reihenfolge nicht der Insertierungsreihenfolge im Single – User entspricht. Der Quellcode der insertierten Pakete ist in Teil 4 nach Paketnummern sortiert.

```
* (INT CONST times, TEXT CONST source) --> TEXT 3-109
** (INT CONST arg, exp) --> INT 8-126
** (REAL CONST a, INT CONST b) --> REAL 26-231
** (REAL CONST b, e) --> REAL 26-222
** (TEXT CONST p, INT CONST x) --> TEXT 15-58
+ (TEXT CONST left, right) --> TEXT 3-103
+ (THESAURUS CONST left, TEXT CONST right) --> THESAURUS 39-47
+ (THESAURUS CONST left, right) --> THESAURUS 39-32
- (TEXT CONST alphabet) --> TEXT 15-43
- (THESAURUS CONST left, TEXT CONST right) --> THESAURUS 39-72
- (THESAURUS CONST left, right) --> THESAURUS 39-57
/ (THESAURUS CONST left, right) --> THESAURUS 39-81
:= (DATASPACE VAR dest, DATASPACE CONST source) 5-23
:= (FILE VAR left, FILE CONST right) 16-160
:= (INITFLAG VAR flag, BOOL CONST flagtrue) 4-39
:= (TASK VAR dest, TASK CONST source) S30-51
:= (THESAURUS VAR dest, THESAURUS CONST source) 13-103
= (TASK CONST left, right) --> BOOL S30-57
```

A

```
abschnitt neu (INT CONST von satznr, bis satznr) 21-2228
ABS (INT CONST argument) --> INT 8-122
abs (INT CONST argument) --> INT 8-114
abs (REAL CONST value) --> REAL 10-329
ABS (REAL CONST value) --> REAL 10-338
aktueller editor --> INT 21-1279
aktueller editor --> INT 21-2158
ALIGN 5-21
alles neu 21-2261
ALL (TASK CONST from) --> THESAURUS S49-649
ALL (TEXT CONST file name) --> THESAURUS 39-96
all --> THESAURUS 14-358
analyze command (TEXT CONST command list, command line,
                INT CONST permitted type, INT VAR command index,
                number of params, TEXT VAR param 1, param 2) 27-117
analyze command (TEXT CONST command list, INT CONST permitted type,
                INT VAR command index, number of params,
                TEXT VAR param 1, param 2) 27-106
AND (INT CONST left, right) --> INT 2-23
ansi cursor (TEXT CONST pre, mid, post) 52-231
any (INT CONST n) --> TEXT 15-64
```

exportierte Objekte alphabetisch geordnet

any (INT CONST n, TEXT CONST a) --> TEXT	15-72
any --> TEXT	15-62
any (TEXT CONST a) --> TEXT	15-70
anything noted --> BOOL	22-689
archive blocks --> INT	48-329
archive (TEXT CONST name)	S49-72
arctand (REAL CONST x) --> REAL	26-218
arctan (REAL CONST y) --> REAL	26-204
at (FILE CONST f, TEXT CONST word) --> BOOL	16-1831
at (TEXT CONST pattern) --> BOOL	22-642

B

baudrate (INT CONST nr, rate)	52-81
begin list	14-237
bildabschnitt neu (INT CONST von zeile, bis zeile)	21-2237
bild neu	21-2249
bild neu (FILE VAR f)	21-2251
bild zeigen	21-2417
bit (INT CONST bits, bit no) --> BOOL	2-35
bits (INT CONST channel, key)	52-89
bits (INT CONST channel, number, parity)	52-85
blockin (DATASPACE VAR ds, INT CONST page nr, code1, code2, INT VAR return code)	5-68
blockin (ROW 256 INT VAR block, INT CONST code1, code2, INT VAR return code)	6-161
block number --> INT	48-43
blockout (DATASPACE CONST ds, INT CONST page nr, code1, code2, INT VAR return code)	5-63
blockout (ROW 256 INT CONST block, INT CONST code1, code2, INT VAR return code)	6-146
bold offset (INT CONST font number) --> INT	S31-563
bound --> TEXT	15-87
bulletin	25-679
bulletin (TEXT CONST packet name)	25-559

C

CA (TEXT CONST old, new)	22-613
cat input (TEXT VAR t, esc char)	6-113
CAT (TEXT VAR result, INT CONST number)	3-374
CAT (TEXT VAR right, TEXT CONST left)	3-99
change all (TEXT CONST old, new)	22-618
change (FILE VAR f, INT CONST from, to, TEXT CONST new)	16-1946
change (TEXT VAR destination, INT CONST from, to, TEXT CONST new)	3-167
change (TEXT VAR destination, TEXT CONST old, new)	3-183
change to (TEXT CONST old, new)	22-596
channel --> INT	6-133
channel (TASK CONST id) --> INT	S30-81
char pitch (INT CONST font number, TEXT CONST char) --> INT	S31-410
check --> BOOL	25-849
check off	25-845
check on	25-841
check read	48-118
check (TEXT CONST name, TASK CONST from)	S49-534
clear error	9-44
clear removed (FILE VAR f)	16-1687
clear (TASK CONST dest)	S49-678

exportierte Objekte alphabetisch geordnet

clock (INT CONST nr) --> REAL	4-62
clock (TASK CONST id) --> REAL	S30-87
close (FILE VAR f)	16-983
code (INT CONST code) --> TEXT	3-53
code (TEXT CONST text) --> INT	3-49
col (FILE CONST f) --> INT	16-1795
col (FILE VAR f, INT CONST new column)	16-1801
col --> INT	22-453
col (INT CONST stelle)	22-448
collect garbage blocks	S43-202
collect heap garbage	3-240
command dialogue --> BOOL	12-36
command dialogue (BOOL CONST status)	12-40
command error	27-247
compress (TEXT CONST text) --> TEXT	3-144
concatenate line (FILE VAR f, BOOL CONST delete blanks)	16-1407
configure	S53-220
CONTAINS (THESAURUS CONST thesaurus, TEXT CONST name) --> BOOL	13-231
continue (INT CONST channel no)	S30-112
continue scan (TEXT CONST scan text)	18-44
control (INT CONST code1, code2, code3, INT VAR return code)	6-142
copy attributes (FILE CONST source file, FILE VAR dest file)	16-1703
copy (DATASPACE CONST source, TEXT CONST dest name)	14-200
copy (TEXT CONST source name, dest name)	14-218
cosd (REAL CONST x) --> REAL	26-170
cos (REAL CONST x) --> REAL	26-162
cout (INT CONST number)	6-128
cover tracks	27-266
cover tracks (TEXT VAR secret)	27-277
create (TEXT CONST name)	14-61
C (TEXT CONST old, new)	22-583
C (TEXT CONST replacement)	22-587
cursor (INT CONST x, y)	6-118
cursor logic (INT CONST dist, modus, TEXT CONST pre, mid, post)	52-237
cursor logic (INT CONST dist, TEXT CONST pre, mid, post)	52-225

D

dataspaces --> INT	S30-121
date (REAL CONST datum) --> TEXT	11-44
date --> TEXT	11-33
date (TEXT CONST datum) --> REAL	11-201
day --> REAL	11-28
day (REAL CONST datum) --> TEXT	11-128
decimal exponent (REAL CONST mantissa) --> INT	10-48
DECR (REAL VAR dest, REAL CONST decrement)	10-393
delete char (TEXT VAR string, INT CONST delete pos)	3-214
delete int (TEXT VAR result, INT CONST delete pos)	3-385
delete record (FILE VAR f)	16-1055
delete (THESAURUS VAR thesaurus, INT CONST index)	13-188
delete (THESAURUS VAR thesaurus, TEXT CONST name, INT VAR index)	13-180
D (INT CONST anz)	22-498
disable stop	9-32
display (TEXT CONST text)	6-81
do command	27-233
do (PROC (TEXT CONST) operate, THESAURUS CONST thesaurus)	39-199
do (PROC (TEXT CONST, TASK CONST) operate, THESAURUS CONST thesaurus, TASK CONST task)	39-163
do (TEXT CONST command)	17-23

exportierte Objekte alphabetisch geordnet

downety (FILE VAR f, TEXT CONST pattern)	16-1884
downety (FILE VAR f, TEXT CONST pattern, INT CONST max line)	16-1890
downety (TEXT CONST muster)	22-557
downety (TEXT CONST muster, INT CONST anz)	22-565
down (FILE VAR f)	16-1071
down (FILE VAR f, INT CONST n)	16-1085
down (FILE VAR f, TEXT CONST pattern)	16-1869
down (FILE VAR f, TEXT CONST pattern, INT CONST max line)	16-1875
down (INT CONST anz)	22-493
down (TEXT CONST muster)	22-513
down (TEXT CONST muster, INT CONST anz)	22-530
ds pages (DATASPACE CONST ds) --> INT	5-51
D (TEXT CONST muster)	22-525

E

edit	22-200
editfile --> FILE	21-2929
edit (FILE VAR f)	22-176
edit (FILE VAR f, INT CONST x, y, x size, y size)	22-186
edit (FILE VAR f, TEXT CONST res, PROC (TEXT CONST) kdc interpreter)	22-193
editget command (BOOL CONST schalter)	21-81
editget (TEXT VAR editsatz)	21-200
editget (TEXT VAR editsatz, INT CONST editlimit, editlaenge)	21-205
editget (TEXT VAR editsatz, INT CONST editlimit, editlaenge, TEXT CONST sep, res, TEXT VAR exit char)	21-87
editget (TEXT VAR editsatz, INT CONST editlimit, TEXT VAR exit char)	21-192
editget (TEXT VAR editsatz, TEXT CONST sep, res, TEXT VAR exit char)	21-196
edit info (FILE CONST f) --> INT	16-1760
edit info (FILE VAR f, INT CONST info)	16-1767
edit (INT CONST i)	22-261
edit (INT CONST i, TEXT CONST res, PROC (TEXT CONST) kommando interpreter)	21-2665
edit (INT CONST von, bis, start, TEXT CONST res, PROC (TEXT CONST) kommando interpreter)	21-2670
edit (TEXT CONST filename)	22-233
edit (TEXT CONST filename, INT CONST x, y, x size, y size)	22-250
edit (THESAURUS CONST nameset)	39-348
elbit cursor	52-247
empty thesaurus --> THESAURUS	13-96
enable stop	9-28
enter incode (INT CONST elan code, TEXT CONST sequenz)	52-211
enter outcode (INT CONST eumelcode, TEXT CONST wert)	52-194
enter outcode (INT CONST eumel code, wartezeit, TEXT CONST sequenz)	52-178
enter outcode (INT CONST eumel code, ziel code)	52-156
enter password (TEXT CONST file name, write pass, read pass)	14-284
enter password (TEXT CONST password)	14-270
eof --> BOOL	22-623
eof (FILE CONST f) --> BOOL	16-1182
erase	S49-308
erase (TEXT CONST file name)	S49-314
erase (TEXT CONST file name, TASK CONST dest)	S49-320
erase (THESAURUS CONST nameset)	39-330
erase (THESAURUS CONST nameset, TASK CONST task)	39-336
e --> REAL	26-26
error code --> INT	9-79
error line --> INT	9-85
error message --> TEXT	9-72
errorstop (INT CONST code, TEXT CONST message)	9-107
errorstop (TEXT CONST message)	9-101

exportierte Objekte alphabetisch geordnet

eumel must advertise -->	SOME	S29-4
exec configuration		S53-447
exec (PROC (TEXT VAR, INT CONST) proc, FILE VAR f, INT CONST i)		16-1853
exec (PROC (TEXT VAR, TEXT CONST) proc, FILE VAR f, TEXT CONST t)		16-1844
exists (TEXT CONST name) -->	BOOL	14-127
exists (TEXT CONST name, TASK CONST from) -->	BOOL	S49-563
exp (REAL CONST z) -->	REAL	26-83
extended char pitch (INT CONST font number, TEXT CONST esc char, char)		S31-430
-->	INT	
extended replacement (INT CONST font number, TEXT CONST esc char, char)		S31-480
-->	TEXT	

F

false -->	BOOL	7-4
fetch all		39-300
fetch all (TASK CONST manager)		39-306
fetch (TEXT CONST file name)		S49-215
fetch (TEXT CONST file name, TASK CONST from)		S49-221
fetch (THESAURUS CONST nameset)		39-276
fetch (THESAURUS CONST nameset, TASK CONST task)		39-282
FILE		16-150
FILLBY (FILE VAR file, THESAURUS CONST thesaurus)		39-254
FILLBY (TEXT CONST file name, THESAURUS CONST thesaurus)		39-267
FILLBY (THESAURUS VAR thesaurus, FILE VAR file)		39-237
fixpoint		S43-208
floor (REAL CONST real) -->	REAL	10-62
flow (INT CONST nr, INT CONST dtype)		52-68
font depth (INT CONST font number) -->	INT	S31-384
font exists (TEXT CONST font name) -->	BOOL	S31-311
font height (INT CONST font number) -->	INT	S31-371
font (INT CONST font number) -->	TEXT	S31-298
font lead (INT CONST font number) -->	INT	S31-358
font string (INT CONST font number) -->	TEXT	S31-537
font table (TEXT CONST new font table)		S31-88
font table -->	TEXT	S31-128
font (TEXT CONST font name) -->	INT	S31-284
forget		14-157
forget (DATASPACE CONST dataspace)		5-31
forget (TEXT CONST name)		14-134
forget (TEXT CONST name, QUIET CONST q)		14-145
forget (THESAURUS CONST nameset)		39-324
format archive (INT CONST format code)		48-378
format (INT CONST format code, TASK CONST dest)		S49-722
format (TASK CONST dest)		S49-740
frac (REAL CONST value) -->	REAL	10-369
FRANGE		16-152

G

getcharety -->	TEXT	21-1208
getchar (TEXT VAR zeichen)		21-1173
get command (TEXT CONST command text)		27-33
get command (TEXT CONST command text, TEXT VAR command line)		27-39
get cursor (INT VAR x, y)		6-124
get editcursor (INT VAR x, y)		21-1219
get editline (TEXT VAR editline, INT VAR editpos, editmarke)		21-917
get (FILE VAR f, INT VAR number)		16-1346

exportierte Objekte alphabetisch geordnet

get (FILE VAR f, REAL VAR number)	16-1354
get (FILE VAR f, TEXT VAR word)	16-1336
get (FILE VAR f, TEXT VAR word, INT CONST max length)	16-1312
get (FILE VAR f, TEXT VAR word, TEXT CONST separator)	16-1268
get font (INT CONST font number, INT VAR indentation pitch, font lead, font height, font depth, ROW 256 INT VAR pitch table)	S31-576
get (INT VAR number)	23-175
getline (FILE VAR f, TEXT VAR text)	16-1152
getline (TEXT VAR textline)	23-203
get list entry (TEXT VAR entry, status text)	14-244
get (REAL VAR number)	23-182
get replacements (INT CONST font number, TEXT VAR replacements, ROW 256 INT VAR replacements table)	S31-595
get secret line (TEXT VAR textline)	23-217
get tabs (FILE CONST f, TEXT VAR tabs)	16-1746
GET (TEXT CONST filename)	22-367
get (TEXT VAR word)	23-130
get (TEXT VAR word, INT CONST length)	23-189
get (TEXT VAR word, TEXT CONST separator)	23-152
get (THESAURUS CONST thesaurus, TEXT VAR name, INT VAR index)	13-281
get window (INT VAR x, y, x size, y size)	21-2935
groesster editor --> INT	21-1281
G (TEXT CONST filename)	22-438

H

headline (FILE CONST f) --> TEXT	16-1732
headline (FILE VAR f, TEXT CONST head)	16-1739
heap size (DATASPACE CONST ds) --> INT	5-43
heap size --> INT	3-236
help (FILE VAR help file)	S40-71
help	S40-61
help (TEXT CONST proc name)	25-328
highest entry (THESAURUS CONST thesaurus) --> INT	13-323
hour --> REAL	11-29
hour --> REAL	11-265

I

id (INT CONST no) --> INT	4-70
incharety (INT CONST time limit) --> TEXT	6-95
incharety --> TEXT	6-91
inchar (TEXT VAR character)	6-87
INCR (REAL VAR dest, REAL CONST increment)	10-387
indentation pitch (INT CONST font number) --> INT	S31-397
INITFLAG	4-19
initialized (INITFLAG VAR flag) --> BOOL	4-50
initialize random (INT CONST start)	8-197
initializerandom (REAL CONST z)	26-263
input buffer size (INT CONST nr, size)	52-76
input (FILE VAR f)	16-962
input --> TRANSPUTDIRECTION	16-851
insert	25-765
insert char (TEXT VAR string, TEXT CONST char, INT CONST insert pos)	3-224
insert int (TEXT VAR result, INT CONST insert pos, number)	3-379
insert record (FILE VAR f)	16-1063
insert (TEXT CONST file name)	25-756
insert (THESAURUS CONST nameset)	39-342

insert (THESAURUS VAR thesaurus, TEXT CONST name)	13-170
insert (THESAURUS VAR thesaurus, TEXT CONST name, INT VAR index)	13-111
int (REAL CONST value) --> INT	10-399
int (TEXT CONST number) --> INT	8-38
is editget --> BOOL	21-913
is error --> BOOL	9-40
is first record (FILE CONST f) --> BOOL	16-1174
is incharety (TEXT CONST muster) --> BOOL	21-1194
is kanji esc (TEXT CONST char) --> BOOL	21-947
is niltask (TASK CONST t) --> BOOL	S30-63
ISUB (TEXT CONST text, INT CONST index) --> INT	3-57

K

ke	4-74
kommando auf taste legen (TEXT CONST taste, kommando)	20-71
kommando auf taste (TEXT CONST taste) --> TEXT	20-79

L

last conversion ok --> BOOL	8-170
last param --> TEXT	12-101
last param (TEXT CONST new)	12-114
len (FILE CONST f) --> INT	16-1930
LENGTH (TEXT CONST text) --> INT	3-124
length (TEXT CONST text) --> INT	3-120
len --> INT	22-443
lernsequenz auf taste legen (TEXT CONST taste, lernsequenz)	20-27
lernsequenz auf taste (TEXT CONST taste) --> TEXT	20-63
LEXEQUAL (TEXT CONST left, right) --> BOOL	3-262
LEXGREATEREQUAL (TEXT CONST left, right) --> BOOL	3-276
LEXGREATER (TEXT CONST left, right) --> BOOL	3-269
lex sort (TEXT CONST dateiname)	16-2025
lex sort (TEXT CONST dateiname, INT CONST sortieranfang)	16-2029
LIKE (TEXT CONST t, pattern) --> BOOL	15-193
LIKE (THESAURUS CONST thesaurus, TEXT CONST pattern) --> THESAURUS	39-142
limit --> INT	22-463
limit (INT CONST limit)	22-458
line	23-94
line (FILE VAR f)	16-1119
line (FILE VAR f, INT CONST lines)	16-1145
line (INT CONST times)	23-103
line no (FILE CONST f) --> INT	16-1197
line no --> INT	22-473
lines (FILE CONST f) --> INT	16-1774
lines --> INT	22-468
line type (FILE CONST f) --> INT	16-1210
line type (FILE VAR f, INT CONST t)	16-1204
link (INT CONST nr, TEXT CONST dtype)	52-142
link (THESAURUS CONST thesaurus, TEXT CONST name) --> INT	13-267
list	24-10
list (FILE VAR f)	24-22
list (FILE VAR list file, TASK CONST from)	S49-588
list fonts	S31-176
list fonts (TEXT CONST name)	S31-164
list font tables	S31-135
list (TASK CONST from)	S49-576
ln (REAL CONST x) --> REAL	26-28

exportierte Objekte alphabetisch geordnet

log10 (REAL CONST x) --> REAL	26-32
log2 (REAL CONST z) --> REAL	26-36
lowest reset (INT CONST bits) --> INT	2-65
lowest set (INT CONST bits) --> INT	2-53

M

margin --> INT	21-2630
margin (INT CONST i)	21-2632
mark --> BOOL	22-628
mark (BOOL CONST mark on)	22-633
mark col (FILE CONST f) --> INT	16-1982
mark (FILE CONST f) --> BOOL	16-1956
mark (FILE VAR f, INT CONST line no, col)	16-1962
mark line no (FILE CONST f) --> INT	16-1973
mark refresh line mode --> BOOL	19-24
mark refresh line mode (BOOL CONST b)	19-28
marksize --> INT	19-13
marksize (INT CONST i)	19-19
matchend (INT CONST x) --> INT	15-97
match (INT CONST x) --> TEXT	15-91
matchpos (INT CONST x) --> INT	15-95
max (INT CONST first, second) --> INT	8-160
maxint --> INT	8-9
max line length (FILE CONST f) --> INT	16-1716
max line length (FILE VAR f, INT CONST new limit)	16-1723
max (REAL CONST a, b) --> REAL	10-375
max real --> REAL	10-40
max text length --> INT	3-35
min (INT CONST first, second) --> INT	8-154
minint --> INT	8-7
min (REAL CONST a, b) --> REAL	10-381
minute --> REAL	11-30
modify (FILE VAR f)	16-976
modify --> TRANSPUTDIRECTION	16-861
MOD (INT CONST left, right) --> INT	8-95
MOD (REAL CONST left, right) --> REAL	10-359
monitor (PROC init system)	S43-40
monitor	S43-34
month (REAL CONST datum) --> TEXT	11-139
myself --> TASK	S30-44

N

name (THESAURUS CONST thesaurus, INT CONST index) --> TEXT	13-274
new configuration	52-39
new (TEXT CONST name) --> DATASPACE	14-86
new type (TEXT CONST dtype)	52-105
next ds page (DATASPACE CONST ds, INT CONST page nr) --> INT	5-59
next larger font exists (INT CONST font number, INT VAR next larger font)	
--> BOOL	S31-318
next smaller font exists (INT CONST font number, INT VAR next smaller font)	
--> BOOL	S31-338
next symbol (FILE VAR f, TEXT VAR symbol)	18-303
next symbol (FILE VAR f, TEXT VAR symbol, INT VAR type)	18-312
next symbol (TEXT VAR symbol)	18-52
next symbol (TEXT VAR symbol, INT VAR type)	18-59
nichts neu	21-2217

nilspace --> DATASPACE	5-27
no do again	17-44
note edit	22-731
note edit (FILE VAR context)	22-701
note file --> FILE	22-695
note (INT CONST number)	22-677
note line	22-683
note (TEXT CONST text)	22-671
no (TEXT CONST question) --> BOOL	12-81
notion (TEXT CONST t, INT CONST r) --> TEXT	15-765
notion (TEXT CONST t) --> TEXT	15-78

O

off string (INT CONST modification) --> TEXT	S31-270
old (TEXT CONST name) --> DATASPACE	14-96
old (TEXT CONST name, INT CONST expected type) --> DATASPACE	14-110
online --> BOOL	6-137
on string (INT CONST modification) --> TEXT	S31-256
open editor (FILE CONST new file, BOOL CONST access)	21-2753
open editor (INT CONST editor nr, FILE CONST new file, BOOL CONST access, INT CONST x start, y, x len start, y len)	21-2792
open editor (INT CONST i)	21-2881
OR (INT CONST left, right) --> INT	2-27
OR (TEXT CONST a, b) --> TEXT	15-54
output (FILE VAR f)	16-969
output --> TRANSPUTDIRECTION	16-856
outsubtext (TEXT CONST source, INT CONST from)	6-39
outsubtext (TEXT CONST source, INT CONST from, to)	6-43
out (TEXT CONST text)	6-35
outtext (TEXT CONST source, INT CONST from, to)	6-47

P

packets	25-701
page	23-112
param position (INT CONST x)	12-95
pattern found --> BOOL	16-674
pause	6-105
pause (INT CONST time limit)	6-100
pcb (INT CONST field) --> INT	4-26
pcb (TASK CONST id, INT CONST field) --> INT	S30-69
pi --> REAL	26-25
pos (FILE CONST f, TEXT CONST pattern, INT CONST i) --> INT	16-1861
pos (TEXT CONST source, low, high, INT CONST from) --> INT	3-140
pos (TEXT CONST source, pattern) --> INT	3-128
pos (TEXT CONST source, pattern, INT CONST from) --> INT	3-132
pos (TEXT CONST source, pattern, INT CONST from, to) --> INT	3-136
prefix (TEXT CONST pattern) --> TEXT	16-546
prot --> BOOL	25-837
prot off	25-832
prot (TEXT CONST file name)	25-826
P (TEXT CONST filename)	22-362
push (TEXT CONST ausfuehrkommando)	21-991
put editline (TEXT CONST editline, INT CONST editpos, editmarke)	21-925
put error	9-116
put (FILE VAR f, INT CONST value)	16-1236
put (FILE VAR f, REAL CONST real)	16-1243

exportierte Objekte alphabetisch geordnet

put (FILE VAR f, TEXT CONST word)	16-1217
put (INT CONST number)	23-73
putline (FILE VAR f, TEXT CONST word)	16-1045
putline (TEXT CONST textline)	23-85
put (REAL CONST number)	23-79
put tabs (FILE VAR f, TEXT CONST tabs)	16-1753
PUT (TEXT CONST filename)	22-293
put (TEXT CONST word)	23-64

Q

QUIET	12-29
quiet --> QUIET	12-31
quit	21-2153
quit last	21-2144

R

random (INT CONST lower bound, upper bound) --> INT	8-208
random --> REAL	26-259
read block (DATASPACE VAR ds, INT CONST ds page no, INT CONST block no, INT VAR return code)	48-268
read (DATASPACE VAR ds)	48-70
read (DATASPACE VAR ds, INT CONST max pages, BOOL CONST error accept)	48-74
read password --> TEXT	14-263
read permission (TEXT CONST name, supply password) --> BOOL	14-312
read record (FILE CONST f, TEXT VAR record)	16-1111
real (INT CONST value) --> REAL	10-424
real (TEXT CONST text) --> REAL	10-243
reinsert (FILE VAR f)	16-1695
release (TASK CONST t)	549-81
remainder --> THESAURUS	39-157
removed lines (FILE CONST f) --> INT	16-1781
remove (FILE VAR f, INT CONST size)	16-1679
rename (TEXT CONST old name, new name)	14-224
rename (THESAURUS VAR thesaurus, INT CONST index, TEXT CONST new)	13-248
rename (THESAURUS VAR thesaurus, TEXT CONST old, new)	13-242
replacement (INT CONST font number, TEXT CONST char) --> TEXT	S31-452
replace (TEXT VAR dest, INT CONST pos, TEXT CONST source)	3-74
replace (TEXT VAR text, INT CONST index, REAL CONST code)	3-69
replace (TEXT VAR text, INT CONST index, value)	3-61
reset bit (INT VAR bits, INT CONST bit no)	2-47
reset (FILE VAR f)	16-926
reset (FILE VAR f, TRANSPUTDIRECTION CONST mode)	16-935
reset range (FILE VAR f)	16-1668
rewind	48-51
rotate (INT VAR bits, INT CONST number of bits)	2-19
round (REAL CONST real, INT CONST digits) --> REAL	10-66
RSUB (TEXT CONST text, INT CONST index) --> REAL	3-65
rubin mode --> BOOL	21-2653
rubin mode (INT CONST editor nr) --> BOOL	21-2655
run	25-744
run again	25-748
run (TEXT CONST file name)	25-735

S

satznr neu	21-2219
satznr zeigen	21-2270
save all	39-312
save all (TASK CONST manager)	39-318
save	S49-411
save system	S43-192
save (TEXT CONST file name)	S49-417
save (TEXT CONST file name, TASK CONST to)	S49-423
save (THESAURUS CONST nameset)	39-288
save (THESAURUS CONST nameset, TASK CONST task)	39-294
say (TEXT CONST message)	12-87
scan (FILE VAR f)	18-296
scan (TEXT CONST scan text)	18-36
search dataspace (INT VAR ds pages)	48-333
second --> REAL	11-31
seek (INT CONST block)	48-47
segments (FILE CONST f) --> INT	16-1788
sequential file (TRANSPUTDIRECTION CONST mode, DATASPACE CONST ds) --> FILE	16-869
sequential file (TRANSPUTDIRECTION CONST mode, TEXT CONST name) --> FILE	16-885
session --> INT	4-22
set bit (INT VAR bits, INT CONST bit no)	2-41
set busy indicator	21-2508
set clock (REAL CONST time)	S43-235
set conversion (BOOL CONST success)	8-174
set date	S43-121
set exp (INT CONST exponent, REAL VAR number)	10-52
set line nr (INT CONST value)	4-34
set marked range (FILE VAR f, FRANGE VAR old range)	16-1993
set range (FILE VAR f, FRANGE VAR new range)	16-1620
set range (FILE VAR f, INT CONST start line, start col, FRANGE VAR old range)	16-1550
setup	S53-453
show	22-283
show (FILE VAR f)	22-266
show (TEXT CONST filename)	22-273
shutup	S43-182
SIGN (INT CONST argument) --> INT	8-110
sign (INT CONST argument) --> INT	8-101
sign (REAL CONST value) --> INT	10-344
SIGN (REAL CONST value) --> INT	10-353
sind (REAL CONST x) --> REAL	26-154
sin (REAL CONST x) --> REAL	26-146
size (INT CONST key) --> INT	48-321
skip dataspace	48-59
small real --> REAL	10-42
somefix (TEXT CONST pattern) --> TEXT	15-123
SOME (TASK CONST task) --> THESAURUS	39-130
SOME (TEXT CONST file name) --> THESAURUS	39-136
SOME (THESAURUS CONST thesaurus) --> THESAURUS	39-105
sort (TEXT CONST dateiname)	16-2016
sort (TEXT CONST dateiname, INT CONST sortieranfang)	16-2020
split line (FILE VAR f, INT CONST split col)	16-1365
split line (FILE VAR f, INT CONST split col, BOOL CONST note indentation)	16-1371
sqrt (REAL CONST z) --> REAL	26-64
status (INT CONST pos, TEXT CONST status pattern)	14-187
status (TASK CONST id) --> INT	S30-75
status (TEXT CONST name, status text)	14-166
status (TEXT CONST name) --> TEXT	14-176

exportierte Objekte alphabetisch geordnet

std kommando interpreter (TEXT CONST taste)	22-58
std tastenbelegung	20-92
std --> TEXT	12-118
stop	9-135
storage (DATASPACE CONST ds) --> INT	5-47
storage info	S40-45
storage (INT VAR size, used)	4-66
storage (TASK CONST id) --> INT	S30-93
stranalyze (ROW 256 INT CONST table, INT VAR sum, INT CONST max sum, TEXT CONST string, INT VAR index, INT CONST to, INT VAR exit code)	3-244
SUB (TEXT CONST text, INT CONST pos) --> TEXT	3-37
subtext (FILE CONST f, INT CONST from, to) --> TEXT	16-1938
subtext (TEXT CONST source, INT CONST from) --> TEXT	3-45
subtext (TEXT CONST source, INT CONST from, to) --> TEXT	3-41
sysin --> TEXT	23-59
sysin (TEXT CONST file name)	23-48
sysout --> TEXT	23-44
sysout (TEXT CONST file name)	23-33

T

tand (REAL CONST x) --> REAL	26-116
tan (REAL CONST x) --> REAL	26-111
TASK	S30-38
task status	S40-34
taste enthaelt kommando (TEXT CONST taste) --> BOOL	20-87
text (INT CONST number, length) --> TEXT	8-25
text (INT CONST number) --> TEXT	8-12
text (REAL CONST real, INT CONST length, fracs) --> TEXT	10-199
text (REAL CONST real, INT CONST length) --> TEXT	10-155
text (REAL CONST real) --> TEXT	10-88
text (TEXT CONST source, INT CONST length, from) --> TEXT	3-95
text (TEXT CONST source, INT CONST length) --> TEXT	3-78
THESAURUS	13-17
time of day (REAL CONST value) --> TEXT	11-170
time of day --> TEXT	11-166
time (REAL CONST value, INT CONST length) --> TEXT	11-178
time (REAL CONST value) --> TEXT	11-174
TIMESOUT (INT CONST times, TEXT CONST text)	6-59
time (TEXT CONST time) --> REAL	11-260
T (INT CONST satz nr)	22-488
to eof (FILE VAR f)	16-1038
to first record (FILE VAR f)	16-1031
to line (FILE VAR f, INT CONST destination line)	16-1023
to line (INT CONST satz nr)	22-478
true --> BOOL	7-4
two bytes --> BOOL	21-952
two bytes (BOOL CONST new mode)	21-954
type (DATASPACE CONST ds) --> INT	5-39
type (DATASPACE CONST ds, INT CONST type)	5-35
type (TEXT CONST ausfuehrkommando)	21-1009

U

ueberschrift neu	21-2221
ueberschrift zeigen	21-2274
U (INT CONST anz)	22-508

UNLIKE (TEXT CONST t, p) --> BOOL	15-191
up (FILE VAR f)	16-1078
up (FILE VAR f, INT CONST n)	16-1091
up (FILE VAR f, TEXT CONST pattern)	16-1899
up (FILE VAR f, TEXT CONST pattern, INT CONST max line)	16-1905
up (INT CONST anz)	22-503
uppety (FILE VAR f, TEXT CONST pattern)	16-1914
uppety (FILE VAR f, TEXT CONST pattern, INT CONST max line)	16-1920
uppety (TEXT CONST muster)	22-570
uppety (TEXT CONST muster, INT CONST anz)	22-578
up (TEXT CONST muster)	22-535
up (TEXT CONST muster, INT CONST anz)	22-552
U (TEXT CONST muster)	22-547

W

warnings --> BOOL	25-861
warnings off	25-857
warnings on	25-853
within kanji (TEXT CONST satz, INT CONST stelle) --> BOOL	21-933
word (FILE CONST f, INT CONST max length) --> TEXT	16-1825
word (FILE CONST f) --> TEXT	16-1809
word (FILE CONST f, TEXT CONST delimiter) --> TEXT	16-1815
word (INT CONST len) --> TEXT	22-656
word --> TEXT	22-646
word (TEXT CONST sep) --> TEXT	22-651
word wrap --> BOOL	21-2623
word wrap (BOOL CONST b)	21-2607
write block (DATASPACE CONST ds, INT CONST ds page no, INT CONST mode, INT CONST block no, INT VAR return code)	48-292
write (DATASPACE CONST ds)	48-134
write (FILE VAR f, TEXT CONST word)	16-1250
write password --> TEXT	14-257
write permission --> BOOL	21-989
write permission (TEXT CONST name, supply password) --> BOOL	14-335
write record (FILE VAR f, TEXT CONST record)	16-1090
write (TEXT CONST word)	23-120

X

XOR (BOOL CONST left, right) --> BOOL	7-7
XOR (INT CONST left, right) --> INT	2-31
xsize --> INT	19-9
xsize (INT CONST i)	19-15
x step conversion (INT CONST steps) --> REAL	S31-229
x step conversion (REAL CONST cm) --> INT	S31-218

Y

year (REAL CONST datum) --> TEXT	11-156
yes (TEXT CONST question) --> BOOL	12-45
y offsets (INT CONST font number) --> TEXT	S31-550
ysize --> INT	19-11
ysize (INT CONST channel, new size, INT VAR old size)	52-72
ysize (INT CONST i)	19-17
y step conversion (INT CONST steps) --> REAL	S31-248
y step conversion (REAL CONST cm) --> INT	S31-237

Z

zeile neu

21-2223

exportierte Objekte alphabetisch geordnet

4. Quellecode der insertierten Pakete

- '(M)' vor der Paketnummer heißt, daß dies Objekt nur im Multi – User vorhanden ist.
- '(S)' vor der Paketnummer heißt, daß dies Objekt nur im Single – User vorhanden ist.
- '(T)' vor der Paketnummer heißt, daß dies Objekt nur in einem System mit Textverarbeitung vorhanden ist.

Die Paketnummer ergibt sich aus der Reihenfolge, in der die Pakete im Multi – User mit Textverarbeitung insertiert wurden. Bitte beachten Sie, daß diese Reihenfolge nicht der Insertierungsreihenfolge im Single – User entspricht. Der Quellecode der insertierten Pakete ist in Teil 4 nach Paketnummern sortiert.

```

1      | "run again impossible"
2      | "recursive run"
3      | "
4      | " Compiler Error : "
5      | "
6      | "
7      | " Fehler entdeckt "
8      | "Keine Fehler gefunden, "
9      | "
10     | " ***** ENDE DER UEBERSETZUNG *****"
11     | "FEHLER bei >> "
12     | " << "
13     | "weiter bei "
14     | "TEXTende (Anfuhrungszeichen) fehlt irgendwo"
15     | "Kommentarende fehlt irgendwo"
16     | "nach dem Hauptprogramm darf kein Paket folgen"
17     | "ungueltiger Name fuer ein DEFINES-Objekt"
18     | "':' fehlt"
19     | "nach ENDPACKET folgt nicht der Paketname"
20     | "ENDPACKET fehlt"
21     | "CONST oder VAR fehlt"
22     | "ungueltiger Name"
23     | "' ' in Deklarationsliste fehlt"
24     | "ist nicht der PROC Name"
25     | "fehlerhaftes Ende des Hauptprogramms"
26     | "ENDPROC fehlt"
27     | "PROC/OP Schachtelung unzuellaessig"
28     | "OP darf kein Parameter sein"
29     | "steht mehrfach im PACKET Interface"
30     | " ist mehrfach deklariert"
31     | "ist schon als Datenobjekt deklariert"
32     | "ist schon als PROC/OP deklariert"
33     | "'' nach Parameterliste erwartet"
34     | "Standard-Schluesselwort kann nicht redefiniert werden"
35     | "ungueltig als BOLD"
36     | "'(' fehlt"
37     | "CONST bzw VAR nicht bei Strukturfeldern"
38     | "'=' fehlt"
39     | "Schluesselwort wird im Paket schon andersartig verwandt"
40     | "Datentyp fehlt"
41     | "ungueltiger OP Name"
42     | "OP muss monadisch oder dyadisch sein"
43     | "ist nicht der OP Name"
44     | "ENDOP fehlt"
45     | "Name nach ENDPROC fehlt"
46     | "Name nach ENDOP fehlt"
47     | "':' fehlt"
48     | "END END ist Unsinn"
49     | "Dieses END... kenne ich nicht"
50     | "ROW Groesse ist kein INT"
51     | "ROW Groesse ist kein Denoter"
52     | "Ein ROW muss mindestens ein Element haben"
53     | "ROW Groesse fehlt"
54     | "Parameter kann man nicht initialisieren"
55     | "Konstanten muessen initialisiert werden"
56     | "':' verwenden"
57     | "''')' fehlt"
58     | "Exponent fehlt"
59     | "Undefinierter Typ"
60     | "Rekursiv definierter Typ"
61     | "Mehrfach definierter Selektor"
62     | "Variable bzw. Abkuerzung in der Paket-Schnittstelle"

```

```

63 |"undefinierte ROW Grosse"
64 |"Typ Deklarationen nur im Pakettrumpf"
65 |"CONST bzw. VAR ohne Zusammenhang"
66 |"ist nicht deklariert, steht aber in der Paket-Schnittstelle"
67 |"ist nicht deklariert"
68 |"unbekanntes Kommando"
69 |"THIS IS NO CORRECT EXTERNAL NUMBER."
70 |"Schluesselwort unzulessig"
71 |"Name erwartet"
72 |"Denoter erwartet"
73 |"ENDPROC ohne Zusammenhang"
74 |"ENDOP ohne Zusammenhang"
75 |"Refinement ohne Zusammenhang"
76 |"Delimiter zwischen Paket-Refinement und Deklaration fehlt"
77 |"unzulessiges Selektor-Symbol (kein Name)"
78 |"BOUND Schachtelungen unzulessig"
79 |"BOUND-Objekte unzulessig als Parameter"
80 |"Textende fehlt"
81 |"TEXT-Denoter zu lang"
82 |
83 |"Denoter-Wert wird fuer diese Maschine zu gross"
84 |"Compiler-Fehler, wenden Sie sich an Ihren Systemberater!"
85 |"ist ein zusammenhangloses Schluesselwort"
86 |"':' nur fuer Initialisierungen, sonst ':'"
87 |"welches Objekt soll verlassen werden?"
88 |"du bist gar nicht innerhalb dieses Refinements"
89 |"nur die eigene PROC / OP kann verlassen werden"
90 |"THEN fehlt"
91 |"FI fehlt"
92 |"BOOL-Ausdruck erwartet"
93 |"ELSE-Teil ist notwendig, da ein Wert geliefert wird"
94 |"INT-Ausdruck erwartet"
95 |"OF fehlt"
96 |"Keine Typanpassung moeglich"
97 |"CASE-Label fehlt"
98 |"mindestens eine CASE-Anweisung geben"
99 |"CASE-Label ist zu gross (skipped)"
100 |"mehrfach definiertes CASE-Label"
101 |"unguetliches Zeichen nach CASE-Label"
102 |"OTHERWISE-Teil fehlt"
103 |"END SELECT fehlt"
104 |"rekursiver Aufruf eines Refinements"
105 |" wird nicht benutzt"
106 |"':' oder Operator ('+', '-', ...) fehlt"
107 |"undefinierter monadischer Operator"
108 |"undefinierter dyadischer Operator"
109 |"Auf die Feinstruktur des Typs kann man nicht mehr zugreifen"
110 |"fuer diesen Typ nicht definierter Selektor"
111 |"INT,REAL,BOOL,TEXT koennen nicht selektiert werden"
112 |"bei ROWs nur Subscription"
113 |"nicht selektierbar"
114 |"unzulessiger Index fuer Subscription"
115 |"[' ohne Zusammenhang"
116 |"'] ohne Zusammenhang"
117 |"']' nach Subscription fehlt"
118 |"unguetlig zwischen Anweisungen"
119 |"nur die letzte Anweisung eines Abschnitts darf einen Wert liefern"
120 |"Der Pakettrumpf kann keinen Wert liefern"
121 |"anstelle des letzten Symbols wurde ein Operand erwartet"
122 |"Der Schleifenrumpf darf keinen Wert liefern"
123 |"die Laufvariable muss eine INT VAR sein"
124 |"wird schon in einer aeusseren Schleife als Laufvariable benutzt"

```

```

125 |"FROM erwartet"
126 |"UPTO bzw DOWNTO fehlt"
127 |"REPEAT fehlt"
128 |"END REP fehlt"
129 |"die Konstante darf nicht veraendert werden"
130 |"in einer FOR-Schleife darf die Laufvariable nicht veraendert werden"
131 |"falscher Typ des Resultats"
132 |"ist CONST, es wird aber ein VAR Parameter verlangt"
133 |"unbekannte Prozedur"
134 |"Parameter-Prozedur liefert falsches Resultat"
135 |"Anzahl bzw. Typen der Parameter sind falsch"
136 |"unbekannte Parameter-Prozedur"
137 |"aktuelle Parameter-Prozedur hat CONST-, formale hat VAR-Parameter"
138 |"Kein Konstruktor moeglich, da die Feinstruktur hier unbekannt ist"
139 |"zu wenig Felder angegeben"
140 |"zu viele Felder angegeben"
141 |"unzulassiger Trenner zwischen Feldern"
142 |"Feld hat falschen Typ"
143 |"falsche Element-Anzahl im ROW-Konstruktor"
144 |"Dieser Typ kann nicht noch mehr konkretisiert werden"
145 |"BOUND-Objekt zu gross"
146 |
147 |"Warnung in Zeile "
148 |" Zeile "
149 |"in Zeile "
150 |" <-----> "
151 |" TYPE undefiniert "
152 |" MODE undefiniert "
153 |"Parameter spezifiziert: "
154 |"Parameter Typ(en) sind: "
155 |" B Code, "
156 |" B Paketdaten generiert"
157 |"Operand: "
158 |"Operanden: "
159 |" "
160 |"erwartet "
161 |"gefunden "
162 |" "
163 |
164 |(* 001 *) END
165 |(* 002 *) ENDPACKET
166 |(* 003 *) ENDOP
167 |(* 004 *) ENDOPERATOR
168 |(* 005 *) ENDPROC
169 |(* 006 *) ENDPROCEDURE

```

```

170 *****|(* 007 *) PACKET
171 |(* 008 *) OP
172 |(* 009 *) OPERATOR
173 |(* 010 *) PROC
174 |(* 011 *) PROCEDURE
175 |(* 012 *) FI
176 |(* 013 *) ENDFI
177 |(* 014 *) ENDREP
178 |(* 015 *) ENDREPEAT
179 |(* 016 *) PER
180 |(* 017 *) ELIF
181 |(* 018 *) ELSE
182 |(* 019 *) UNTIL
183 |(* 020 *) CASE
184 |(* 021 *) OTHERWISE
185 |(* 022 *) ENDSELECT
186 |(* 023 *) INTERNAL
187 |(* 024 *) DEFINES
188 |(* 025 *) LET
189 |(* 026 *) TYPE
190 |(* 027 *) INT
191 |(* 028 *) REAL
192 |(* 029 *) DATASPACE
193 |(* 030 *) TEXT
194 |(* 031 *) BOOL
195 |(* 032 *) BOUND
196 |(* 033 *) ROW
197 |(* 034 *) STRUCT
198 |(* 035 *) CONST
199 |(* 036 *) VAR
200 |(* 037 INIT CONTROL *) INTERNAL
201 |(* 038 *) CONCR
202 |(* 039 *) REP
203 |(* 040 *) REPEAT
204 |(* 041 *) SELECT
205 |(* 042 *) EXTERNAL
206 |(* 043 *) IF
207 |(* 044 *) THEN
208 |(* 045 *) OF
209 |(* 046 *) FOR
210 |(* 047 *) FROM
211 |(* 048 *) UPTO
212 |(* 049 *) DOWNTO
213 |(* 050 *) WHILE
214 |(* 051 *) LEAVE
215 |(* 052 *) WITH
216 |(* 053 *) TRUE
217 |(* 054 *) FALSE
218 |(* 055 *) :: SBL := INCR DECR
219 |(* 056 *) + - * / DIV MOD
220 |
221 |
222 |
223 |
224 |
225 |
226 |
227 |(*040 *) MAIN
228 |(*043*) ENDOFFILE
229 |

```

```
230 a *****|PACKET a :
231          |
232 out .....|PROC out (TEXT CONST t) :
233          |   INTERNAL 60
234          |   ENDPROC out ;
235          |
236 outtext .....|PROC out text (TEXT CONST t, INT CONST typ) :
237          |   INTERNAL 257 ;
238          |   IF typ = typ
239          |     THEN out (t)
240          |   FI
241          |   ENDPROC out text ;
242          |
243 outline .....|PROC out line (INT CONST typ) :
244          |   INTERNAL 258 ;
245          |   IF typ = typ
246          |     THEN out ("13"10")
247          |   FI
248          |   ENDPROC out line ;
249          |
250          |   ENDPACKET a ;
251          |
252          |
253          |
254          |
255          |
256          |
257          |
258          |
259          |
260          |
261          |
262          |
```



```

47 resetbit .....|PROC reset bit (INT VAR bits,INT CONST bit no) :
48                |
49                |   bits := bits XOR (bits AND bit mask (bit no+1))
50                |
51                |ENDPROC reset bit ;
52                |

53 lowestset .....|INT PROC lowest set (INT CONST bits) :
54                |
55                |   INT VAR mask index ;
56                |   FOR mask index FROM 1 UPTO 16 REP
57                |     IF (bits AND bit mask (mask index)) <> 0
58                |       THEN LEAVE lowest set WITH mask index - 1
59                |     FI
60                |   PER ;
61                |   -1
62                |
63                |ENDPROC lowest set ;
64                |

65 lowestreset .....|INT PROC lowest reset (INT CONST bits) :
66                |
67                |   INT VAR mask index ;
68                |   FOR mask index FROM 1 UPTO bits per int REP
69                |     IF (bits AND bit mask (mask index)) = 0
70                |       THEN LEAVE lowest reset WITH mask index - 1
71                |     FI
72                |   PER ;
73                |   -1
74                |
75                |ENDPROC lowest reset ;
76                |
77                |ENDPACKET bits ;

```

```

1      |(* ----- VERSION 3      06.03.86 ----- *)
2  text *****|PACKET text DEFINES
3
4      |          max text length ,
5      |          SUB ,
6      |          subtext ,
7      |          text ,
8      |          length , LENGTH ,
9      |          CAT ,
10     |          * ,
11     |          * ,
12     |          replace ,
13     |          change ,
14     |          change all ,
15     |          compress ,
16     |          pos .
17     |          code ,
18     |          ISUB ,
19     |          RSUB ,
20     |          delete char ,
21     |          insert char ,
22     |          delete int ,
23     |          insert int ,
24     |          heap size ,
25     |          collect heap garbage ,
26     |          stranalyze ,
27     |          LEXEQUAL ,
28     |          LEXGREATER ,
29     |          LEXGREATEREQUAL :
30
31
32
33     |TEXT VAR text buffer , tail buffer ;
34
35     |INT CONST max text length := 32000 ;
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

```

```

53 code .....|TEXT PROC code (INT CONST code) :
54           |EXTERNAL 47
55           |ENDPROC code ;
56           |

57 ISUB .....|INT OP ISUB (TEXT CONST text, INT CONST index) :
58           |EXTERNAL 44
59           |ENDOP ISUB ;
60           |

61 replace .....|PROC replace (TEXT VAR text, INT CONST index, value) :
62           |EXTERNAL 45
63           |ENDPROC replace ;
64           |

65 RSUB .....|REAL OP RSUB (TEXT CONST text, INT CONST index) :
66           |EXTERNAL 100
67           |ENDOP RSUB ;
68           |

69 replace .....|PROC replace (TEXT VAR text, INT CONST index, REAL CONST code) :
70           |EXTERNAL 101
71           |ENDPROC replace ;
72           |
73           |

74 replace .....|PROC replace (TEXT VAR dest, INT CONST pos, TEXT CONST source) :
75           |EXTERNAL 51
76           |ENDPROC replace ;
77           |

78 text .....|TEXT PROC text (TEXT CONST source, INT CONST length) :
79           |
80           |    IF length < LENGTH source
81           |    THEN text buffer := subtext (source,1,length)
82           |    ELSE text buffer := source ;
83           |         mit blanks auffuellen
84           |
85           |    FI ;
86           |    text buffer .

87 mitblanksauffuellen |mit blanks auffuellen :
88           |INT VAR i ;
89           |FOR i FROM 1 UPTO length - LENGTH source REP
90           |    text buffer CAT " "
91           |PER .
92           |
93           |ENDPROC text ;
94           |

95 text .....|TEXT PROC text (TEXT CONST source, INT CONST length, from) :
96           |    text ( subtext (source, from) , length )
97           |ENDPROC text ;
98           |

```

```

99 CAT .....|OP CAT (TEXT VAR right, TEXT CONST left) :
100           |EXTERNAL 52
101           |ENDOP CAT ;
102           |

103 + .....|TEXT OP + (TEXT CONST left, right) :
104           |text buffer := left ;
105           |text buffer CAT right ;
106           |text buffer
107           |ENDOP + ;
108           |

109 * .....|TEXT OP * (INT CONST times, TEXT CONST source) :
110           |
111           |text buffer := "" ;
112           |INT VAR i ;
113           |FOR i FROM 1 UPTO times REP
114           |text buffer CAT source
115           |PER ;
116           |text buffer
117           |
118           |ENDOP * ;
119           |

120 length .....|INT PROC length (TEXT CONST text) :
121           |EXTERNAL 53
122           |ENDPROC length ;
123           |

124 LENGTH .....|INT OP LENGTH (TEXT CONST text) :
125           |EXTERNAL 53
126           |ENDOP LENGTH ;
127           |

128 pos .....|INT PROC pos (TEXT CONST source, pattern) :
129           |EXTERNAL 54
130           |ENDPROC pos ;
131           |

132 pos .....|INT PROC pos (TEXT CONST source, pattern, INT CONST from) :
133           |EXTERNAL 55
134           |ENDPROC pos ;
135           |

136 pos .....|INT PROC pos (TEXT CONST source, pattern, INT CONST from, to) :
137           |EXTERNAL 56
138           |ENDPROC pos ;
139           |

140 pos .....|INT PROC pos (TEXT CONST source, low, high, INT CONST from) :
141           |EXTERNAL 58
142           |ENDPROC pos ;
143           |

```

```

144 compress .....|TEXT PROC compress (TEXT CCNST text) :
145                |
146                |   INT VAR begin, end ;
147                |
148                |   search first non blank ;
149                |   search last non blank ;
150                |   text buffer := subtext (text, begin, end) ;
151                |   text buffer .
152                |
153 searchfirstnonblank |search first non blank :
154                |   begin := 1 ;
155                |   WHILE (text SUB begin) = " " REP
156                |     begin INCR 1
157                |   PER .
158                |
159 searchlastnonblank |search last non blank :
160                |   end := LENGTH text ;
161                |   WHILE (text SUB end) = " " REP
162                |     end DECR 1
163                |   PER .
164                |
165                |ENDPROC compress ;
166                |
167 change .....|PROC change (TEXT VAR destination, INT CONST from, to, TEXT CONST
+                |   new) :
168                |
169                |   IF LENGTH new = to - from + 1 AND to <= LENGTH destination
170                |     THEN replace (destination, from, new)
171                |     ELSE change via buffer
172                |   FI .
173                |
174 changeviabuffer |change via buffer :
175                |   text buffer := subtext (destination, 1, from-1) ;
176                |   text buffer CAT new ;
177                |   tail buffer := subtext (destination, to + 1) ;
178                |   text buffer CAT tail buffer ;
179                |   destination := text buffer
180                |
181                |ENDPROC change ;
182                |
183 change .....|PROC change (TEXT VAR destination, TEXT CONST old, new) :
184                |
185                |   INT CONST position := pos (destination, old) ;
186                |   IF position > 0
187                |     THEN change (destination, position, position + LENGTH old -1,
+                |       new)
188                |   FI
189                |
190                |ENDPROC change ;
191                |
192 changeall .....|PROC change all (TEXT VAR destination, TEXT CONST old, new) :
193                |
194                |   INT VAR position := pos (destination, old) ;

```

```

195      | IF LENGTH old = LENGTH new
196      | THEN change by replace
197      | ELSE change by change
198      | FI .
199
200  changebyreplace | change by replace :
201      | WHILE position > 0 REP
202      |   replace (destination, position, new) ;
203      |   position := pos (destination, old, position + LENGTH new)
204      | PER .
205
206  changebychange   | change by change :
207      | WHILE position > 0 REP
208      |   change (destination, position, position + LENGTH old - 1 , new)
209      |   position := pos (destination, old, position + LENGTH new)
210      | PER .
211
212      |ENDPROC change all ;
213
214  deletechar .....|PROC delete char (TEXT VAR string, INT CONST delete pos) :
215      |
216      | IF delete pos > 0
217      | THEN tail buffer := subtext (string, delete pos + 1) ;
218      |   string := subtext (string, 1, delete pos - 1) ;
219      |   string CAT tail buffer
220      |
221      | FI
222      |END PROC delete char ;
223
224  insertchar .....|PROC insert char (TEXT VAR string, TEXT CONST char,
225                  | INT CONST insert pos) :
226      |
227      | IF insert pos > 0 AND insert pos <= LENGTH string + 1
228      | THEN tail buffer := subtext (string, insert pos) ;
229      |   string := subtext (string, 1, insert pos - 1) ;
230      |   string CAT char ;
231      |   string CAT tail buffer
232      |
233      | FI
234      |END PROC insert char ;
235
236  heapsize .....  |INT PROC heap size :
237      | EXTERNAL 93
238      |ENDPROC heap size ;
239
240  collectheapgarbage .....|PROC collect heap garbage :
241      | EXTERNAL 94
242      |ENDPROC collect heap garbage ;
243

```

```

244 stranalyze .....|PROC stranalyze (ROW 256 INT CONST table, INT VAR sum, INT CONST max
+                   |    sum,
245                   |                TEXT CONST string, INT VAR index, INT CONST to,
246                   |                INT VAR exit code) :
247                   |    EXTERNAL 57
248                   |ENDPROC stranalyze ;
249
250                   |{*****}
251                   |(* lexikographische Vergleiche *)
252                   |(* Nach DIN 5007, Abschnitt 1 und Abschnitt 3.2 (Bindestrich) *)
253                   |(* Autor: Rainer Hahn, Jochen Liedtke *)
254                   |(* Stand: 1.7.4 (Jan. 1985) *)
255                   |{*****}
256                   |LET first umlaut = ""214"" ,
257                   |    umlauts      = ""214""215""216""217""218""219""251"" ;
258
259                   |
260                   |TEXTI VAR left letter, right letter;
261

```

```

262 LEXEQUAL .....|BOOL OP LEXEQUAL (TEXT CONST left, right) :
263
264                   |    compare (left, right) ;
265                   |    left letter = right letter
266
267                   |ENDOP LEXEQUAL ;
268

```

```

269 LEXGREATER .....|BOOL OP LEXGREATER (TEXT CONST left, right) :
270
271                   |    compare (left, right) ;
272                   |    left letter > right letter
273
274                   |ENDOP LEXGREATER ;
275

```

```

276 LEXGREATEREQUAL .....|BOOL OP LEXGREATEREQUAL (TEXT CONST left, right) :
277
278                   |    compare (left, right) ;
279                   |    left letter >= right letter
280
281                   |ENDOP LEXGREATEREQUAL ;
282

```

```

283 compare .....|PROC compare (TEXT CONST left, right) :
284
285                   |    to begin of lex relevant text ;
286                   |    REP
287                   |        get left letter ;
288                   |        get right letter
289                   |    UNTIL NOT letter match OR both ended PER .
290

```

```

291 tobeginoflexrelevantte |to begin of lex relevant text :
292                   |    INT VAR
293                   |    left pos := pos (left, ""65"", ""254"", 1) ,
294                   |    right pos := pos (right, ""65"", ""254"", 1) ;
295                   |    IF left pos = 0

```

```

296          THEN left pos := LENGTH left + 1
297          FI ;
298          IF right pos = 0
299          THEN right pos := LENGTH right + 1
300          FI .
301
302  getleftletter  get left letter :
303                left letter := left SUB left pos ;
304                left pos INCR 1 .
305
306  getrightletter get right letter :
307                right letter := right SUB right pos ;
308                right pos INCR 1 .
309
310  lettermatch   letter match :
311                IF left letter = right letter
312                THEN TRUE
313                ELSE dine (left, left letter, left pos) ;
314                         dine (right, right letter, right pos) ;
315                         IF exactly one letter is double letter
316                         THEN expand other letter
317                         FI ;
318                         left letter = right letter
319                FI .
320
321  exactlyoneletterisdoub exactly one letter is double letter :
322                LENGTH left letter <> LENGTH right letter.
323
324  expandotherletter expand other letter :
325                IF LENGTH left letter = 1
326                THEN left letter CAT (left SUB left pos) ;
327                         left pos INCR 1
328                ELSE right letter CAT (right SUB right pos) ;
329                         right pos INCR 1
330                FI .
331
332  bothended     both ended : left letter = "" .
333
334  ENDPROC compare ;
335
336  dine ..... PROC dine (TEXT CONST string, TEXT VAR char, INT VAR string pos)
337
338                skip non letter chars ;
339                IF is capital letter
340                THEN translate to small letter
341                ELIF char >= first umlaut
342                THEN translate umlaut
343                FI .
344
345  skipnonletterchars skip non letter chars :
346                WHILE NOT (is letter OR end of string) REP
347                char := string SUB string pos ;
348                string pos INCR 1

```

```

349          | PER .
350
351  translate to small letter | translate to small letter :
352          |   char := code (code (char) + 32) .
353
354  translate umlaut         | translate umlaut :
355          |   SELECT pos (umlauts, char) OF
356          |     CASE 1,4 : char := "ae"
357          |     CASE 2,5 : char := "oe"
358          |     CASE 3,6 : char := "ue"
359          |     CASE 7   : char := "ss"
360          |   ENDSELECT .
361
362  is capital letter        | is capital letter :
363          |   INT VAR char code := code (char) ;
364          |   65 <= char code AND char code <= 90 .
365
366  is letter                | is letter :
367          |   char code := code (char) OR 32 ;
368          |   (97 <= char code AND char code <= 122) OR char code >= 128 .
369
370  end of string            | end of string : char = "" .
371
372          | ENDPROC dine ;
373
374  CAT .....| OP CAT (TEXT VAR result, INT CONST number) :
375          |   result CAT " ";
376          |   replace (result, LENGTH result DIV 2, number);
377          | END OP CAT;
378
379  insert int .....| PROC insert int (TEXT VAR result, INT CONST insert pos, number)
380          |   INT VAR pos := insert pos * 2 - 1;
381          |   change (result, pos, pos - 1, " ");
382          |   replace (result, insert pos, number);
383          | END PROC insert int;
384
385  delete int .....| PROC delete int (TEXT VAR result, INT CONST delete pos) :
386          |   INT VAR pos := delete pos * 2;
387          |   change (result, pos - 1, pos, "")
388          | END PROC delete int;
389          |
390          | ENDPACKET text ;

```

```

1
2  pcbandinitcontrol *****|PACKET pcb and init control DEFINES      (* Autor: J.Liedtke *)
3                               |                                                (* Stand: 25.08.84 *)
4                               |  session ,
5                               |  pcb ,
6                               |  set line nr ,
7                               |  clock ,
8                               |  INITFLAG ,
9                               |  := ,
10                              |  initialized ,
11                              |  storage ,
12                              |  id ,
13                              |  ke :
14
15
16                              |LET line number field      = 1 ,
17                              |  myself id field        = 9 ;
18
19                              |TYPE INITFLAG = INT ;
20
21
22 session .....|INT PROC session :
23                |  EXTERNAL 126
24                |ENDPROC session ;
25
26 pcb .....|INT PROC pcb (INT CONST field) :
27            |  EXTERNAL 80
28            |ENDPROC pcb ;
29
30 writepcb .....|PROC write pcb (INT CONST task nr, field, value) :
31              |  EXTERNAL 105
32              |ENDPROC write pcb ;
33
34 setlinenr .....|PROC set line nr (INT CONST value) :
35                |  write pcb (pcb (myself id field), line number field, value)
36                |ENDPROC set line nr ;
37
38
39 := .....|OP := (INITFLAG VAR flag, BOOL CONST flagtrue) :
40
41          |  IF flagtrue
42          |    THEN CONCR (flag) := myself no
43          |    ELSE CONCR (flag) := 0
44          |  FI .
45
46 myselfno .....|myself no :  pcb (myself id field) AND 255 .
47
48                |ENDOP := ;
49

```

```

50 initialized .....|BOOL PROC initialized (INITFLAG VAR flag) :
51                   |
52                   |   IF CONCR (flag) = myself no
53                   |   THEN TRUE
54                   |   ELSE CONCR (flag) := myself no ;
55                   |   FALSE
56                   |   FI .
57                   |
58   myselfno        |myself no :   pcb (myself id field) AND 255 .
59                   |
60                   |ENDPROC initialized ;
61                   |

62 clock .....|REAL PROC clock (INT CONST nr) :
63                   |   EXTERNAL 102
64                   |ENDPROC clock ;
65                   |

66 storage .....|PROC storage (INT VAR size, used) :
67                   |   EXTERNAL 89
68                   |ENDPROC storage ;
69                   |

70 id .....|INT PROC id (INT CONST no) :
71                   |   EXTERNAL 129
72                   |ENDPROC id ;
73                   |

74 ke .....|PROC ke :
75                   |   EXTERNAL 6
76                   |ENDPROC ke ;
77                   |
78                   |ENDPACKET pcb and init control ;

```

```

1      | (* ----- VERSION 3      22.04.86 -----
2  dataspace *****| PACKET dataspace DEFINES
3
4      | := ,
5      | nilspace ,
6      | forget ,
7      | type ,
8      | heap size ,
9      | storage ,
10     | ds pages ,
11     | next ds page ,
12     | blackout ,
13     | blockin ,
14     | ALIGN :
15
16
17     | LET myself id field = 9 ,
18     | lowest ds number = 4 ,
19     | highest ds number = 255 ;
20
21     | TYPE ALIGN = ROW 252 INT ;
22
23 := .....| OP := (DATASPACE VAR dest, DATASPACE CONST source ) :
24     | EXTERNAL 70
25     | ENDOP := ;
26
27 nilspace .....| DATASPACE PROC nilspace :
28     | EXTERNAL 69
29     | ENDPROC nilspace ;
30
31 forget .....| PROC forget (DATASPACE CONST dataspace ) :
32     | EXTERNAL 71
33     | ENDPROC forget ;
34
35 type .....| PROC type (DATASPACE CONST ds, INT CONST type) :
36     | EXTERNAL 72
37     | ENDPROC type ;
38
39 type .....| INT PROC type (DATASPACE CONST ds) :
40     | EXTERNAL 73
41     | ENDPROC type ;
42
43 heapsize .....| INT PROC heap size (DATASPACE CONST ds) :
44     | EXTERNAL 74
45     | ENDPROC heap size ;
46

```

```
47 storage .....|INT PROC storage (DATASPACE CONST ds) :
48                |  (ds pages (ds) + 1) DIV 2
49                |ENDPROC storage ;
50                |

51 dspages .....|INT PROC ds pages (DATASPACE CONST ds) :
52                |  pages (ds, pcb (myself id field))
53                |ENDPROC ds pages ;
54                |

55 pages .....|INT PROC pages (DATASPACE CONST ds, INT CONST task nr) :
56                |  EXTERNAL 88
57                |ENDPROC pages ;
58                |

59 nextdspage .....|INT PROC next ds page (DATASPACE CONST ds, INT CONST page nr) :
60                |  EXTERNAL 87
61                |ENDPROC next ds page ;
62                |

63 blackout .....|PROC blackout (DATASPACE CONST ds, INT CONST page nr, code1, code2,
64                |                INT VAR return code) :
65                |  EXTERNAL 85
66                |ENDPROC blackout ;
67                |

68 blockin .....|PROC blockin (DATASPACE VAR ds, INT CONST page nr, code1, code2,
69                |                INT VAR return code) :
70                |  EXTERNAL 86
71                |ENDPROC blockin ;
72                |
73                |ENDPACKET dataspace ;
```

```

1
2 basictransput *****|PACKET basic transput DEFINES
3                       |out ,
4                       |outsubtext ,
5                       |outtext ,
6                       |TIMESOUT ,
7                       |cout ,
8                       |display ,
9                       |inchar ,
10                      |incharety ,
11                      |cat input ,
12                      |pause ,
13                      |cursor ,
14                      |get cursor ,
15                      |channel ,
16                      |online ,
17                      |control ,
18                      |blockout ,
19                      |blockin :
20
21
22
23                      |LET channel field = 4 ,
24                      |    blank times 64 =
25                      |    "
26                      |    " ;
27
28                      |LET BLOCKIO = STRUCT (ALIGN page align, ROW 256 INT buffer) ,
29                      |    buffer page = 2 ;
30
31                      |BOUND BLOCKIO VAR block io ;
32                      |DATASPACE VAR block io ds ;
33                      |INITFLAG VAR this packet := FALSE ;
34
35 out .....|PROC out (TEXT CONST text) :
36           |EXTERNAL 60
37           |ENDPROC out ;
38
39 outsubtext .....|PROC outsubtext ( TEXT CONST source, INT CONST from) :
40           |EXTERNAL 62
41           |END PROC outsubtext;
42
43 outsubtext .....|PROC outsubtext (TEXT CONST source, INT CONST from, to) :
44           |EXTERNAL 63
45           |END PROC outsubtext;
46
47 outtext .....|PROC outtext ( TEXT CONST source, INT CONST from, to) :
48           |out subtext (source, from, to) ;
49           |INT VAR trailing ;
50           |IF from <= LENGTH source
51           |    THEN trailing := to - LENGTH source
52           |    ELSE trailing := to + 1 - from
53           |FI ;

```

```

54          | IF trailing > 0
55          | THEN trailing TIMEOUT " "
56          | FI
57          |ENDPROC outtext ;
58          |

59 TIMEOUT .....|OP TIMEOUT (INT CONST times, TEXT CONST text) :
60          |
61          | IF text = " "
62          | THEN fast timeout blank
63          | ELSE timeout
64          | FI .
65          |

66 fasttimeoutblank |fast timeout blank :
67          | INT VAR i := 0 ;
68          | WHILE i + 64 < times REP
69          |   out (blank times 64) ;
70          |   i INCR 64
71          | PER ;
72          | outsubtext (blank times 64, 1, times - i) .
73          |

74 timeout          |timeout :
75          | FOR i FROM 1 UPTO times REP
76          |   out(text)
77          | ENDREP .
78          |
79          |ENDOP TIMEOUT ;
80          |

81 display .....|PROC display (TEXT CONST text) :
82          | IF online
83          | THEN out (text)
84          | FI
85          |ENDPROC display ;
86          |

87 inchar .....|PROC inchar (TEXT VAR character) :
88          | EXTERNAL 64
89          |ENDPROC inchar ;
90          |

91 incharety .....|TEXT PROC incharety :
92          | EXTERNAL 65
93          |END PROC incharety ;
94          |

95 incharety .....|TEXT PROC incharety (INT CONST time limit) :
96          | internal pause (time limit) ;
97          | incharety
98          |ENDPROC incharety ;
99          |

```

```

100 pause .....|PROC pause (INT CONST time limit) :
101             | internal pause (time limit) ;
102             | TEXT CONST dummy := incharety
103             |ENDPROC pause ;
104             |

105 pause .....|PROC pause :
106             | TEXT VAR dummy; inchar (dummy)
107             |ENDPROC pause ;
108             |

109 internalpause .....|PROC internal pause (INT CONST time limit) :
110             | EXTERNAL 66
111             |ENDPROC internal pause ;
112             |

113 catinput .....|PROC cat input (TEXT VAR t, esc char) :
114             | EXTERNAL 68
115             |ENDPROC cat input ;
116             |
117             |

118 cursor .....|PROC cursor (INT CONST x, y) :
119             | out ("6");
120             | out (code(y-1)) ;
121             | out (code(x-1)) ;
122             |ENDPROC cursor ;
123             |

124 getcursor .....|PROC get cursor (INT VAR x, y) :
125             | EXTERNAL 67
126             |ENDPROC get cursor ;
127             |

128 cout .....|PROC cout (INT CONST number) :
129             | EXTERNAL 61
130             |ENDPROC cout ;
131             |
132             |

133 channel .....|INT PROC channel :
134             | pcb (channel field)
135             |ENDPROC channel ;
136             |

137 online .....|BOOL PROC online :
138             | pcb (channel field) <> 0
139             |ENDPROC online ;
140             |
141             |

```

```

142 control .....|PROC control (INT CONST code1, code2, code3, INT VAR return code)
143                |EXTERNAL 84
144                |ENDPROC control ;
145                |

146 blackout .....|PROC blackout (ROW 256 INT CONST block, INT CONST code1, code2,
147                |          INT VAR return code) :
148                |
149                |   access block io ds ;
150                |   block io.buffer := block ;
151                |   blackout (block io ds, buffer page, code1, code2, return code)
152                |

153   accessblockiods   |access block io ds :
154                |   IF NOT initialized (this packet)
155                |       THEN block io ds := nilspace
156                |   FI ;
157                |   block io := block io ds .
158                |
159                |ENDPROC blackout ;
160                |

161 blockin .....|PROC blockin (ROW 256 INT VAR block, INT CONST code1, code2,
162                |          INT VAR return code) :
163                |
164                |   access block io ds ;
165                |   blockin (block io ds, buffer page, code1, code2, return code) ;
166                |   block := block io.buffer .
167                |

168   accessblockiods   |access block io ds :
169                |   IF NOT initialized (this packet)
170                |       THEN block io ds := nilspace
171                |   FI ;
172                |   block io := block io ds .
173                |
174                |ENDPROC blockin ;
175                |
176                |ENDPACKET basic transput ;
177                |
178                |

```

```
1
2 bool *****|PACKET bool DEFINES XOR, true, false :
3
4               |BOOL CONST true := TRUE ,
5               |         false:= FALSE ;
6               |
7 XOR .....|BOOL OP XOR (BOOL CONST left, right) :
8
9           |   IF left THEN NOT right
10          |       ELSE right
11          |   FI
12
13          |ENDOP XOR ;
14
15          |ENDPACKET bool ;
```

```

1      | (* ----- STAND :      23.10.85
+      | -----*)
2 integer ***** PACKET integer DEFINES text, int, MOD,
3      | sign, SIGN, abs, ABS, **, min, max, minint,
+      | maxint,
4      | random, initialize random ,
5      | last conversion ok, set conversion :
6      |
7 minint .....| INT PROC minint : -32767 - 1 ENDPROC minint ;
8      |
9 maxint .....| INT PROC maxint : 32767 ENDPROC maxint ;
10     |
11     |
12 text .....| TEXT PROC text (INT CONST number) :
13     |
14     | IF number = minint THEN "-32768"
15     | ELIF number < 0 THEN "-" + text(-number)
16     | ELIF number <= 9 THEN code (number + 48)
17     | ELSE text (number DIV 10) + digit
18     | FI .
19     |
20 digit .....| digit :
21     | code ( number MOD 10 + 48 ) .
22     |
23     | ENDPROC text ;
24     |
25 text .....| TEXT PROC text (INT CONST number, length) :
26     |
27     | TEXT VAR result := text (number) ;
28     | INT CONST number length := LENGTH result ;
29     | IF number length < length
30     | THEN (length - number length) * " " + result
31     | ELIF number length > length
32     | THEN length * "*"
33     | ELSE result
34     | FI
35     |
36     | ENDPROC text ;
37     |
38 int .....| INT PROC int (TEXT CONST number) :
39     |
40     | skip blanks and sign ;
41     | get value ;
42     | result .
43     |
44 skipblanksandsign .....| skip blanks and sign :
45     | BOOL VAR number is positive ;
46     | INT VAR pos := 1 ;
47     | skip blanks ;
48     | IF (number SUB pos) = "-"

```

```

49         THEN number is positive := FALSE ;
50         pos INCR 1
51     ELIF (number SUB pos) = "+"
52         THEN number is positive := TRUE ;
53         pos INCR 1
54     ELSE number is positive := TRUE
55     FI .
56
57     getvalue      get value :
58                   INT VAR value ;
59                   get first digit ;
60                   WHILE is digit REP
61                       value := value * 10 + digit ;
62                       pos INCR 1
63                   PER ;
64                   set conversion ok result .
65
66     getfirstdigit get first digit :
67                   IF is digit
68                       THEN value := digit ;
69                       pos INCR 1
70                   ELSE set conversion (FALSE) ;
71                       LEAVE int WITH 0
72                   FI .
73
74     isdigit      is digit : 0 <= digit AND digit <= 9 .
75
76     digit        digit : code (number SUB pos) - 48 .
77
78     result       result :
79                   IF number is positive
80                       THEN value
81                   ELSE - value
82                   FI .
83
84     setconversionokresult set conversion ok result :
85                   skip blanks ;
86                   conversion ok := (pos > LENGTH number) .
87
88     skipblanks   skip blanks :
89                   WHILE (number SUB pos) = " " REP
90                       pos INCR 1
91                   PER .
92
93                   ENDPROC int ;
94
95     MOD ..... INT OP MOD (INT CONST left, right) :
96
97                   EXTERNAL 43
98
99                   ENDP MOD ;
100

```

```

101 sign .....|INT PROC sign (INT CONST argument) :
102           |
103           |   IF argument < 0 THEN -1
104           |   ELIF argument > 0 THEN 1
105           |   ELSE 0
106           |   FI
107           |
108           |ENDPROC sign ;
109           |

```

```

110 SIGN .....|INT OP SIGN (INT CONST argument) :
111           |   sign (argument)
112           |ENDOP SIGN ;
113           |

```

```

114 abs .....|INT PROC abs (INT CONST argument) :
115           |
116           |   IF argument > 0 THEN argument
117           |   ELSE - argument
118           |   FI
119           |
120           |ENDPROC abs ;
121           |

```

```

122 ABS .....|INT OP ABS (INT CONST argument) :
123           |   abs (argument)
124           |ENDOP ABS ;
125           |

```

```

126 ** .....|INT OP ** (INT CONST arg, exp) :
127           |
128           |   INT VAR x := arg , z := 1 ,
129           |   counter := exp ;
130           |
131           |   IF exp = 0
132           |   THEN LEAVE ** WITH 1
133           |   ELIF exp < 0
134           |   THEN LEAVE ** WITH 1 DIV arg
135           |   FI ;
136           |
137           |   WHILE counter >= 2 REP
138           |   calculate new x and z ;
139           |   counter := counter DIV 2 ;
140           |   ENDREP ;
141           |   z * x .
142           |

```

```

143 calculatenewxandz calculate new x and z :
144           |   IF counter is not even
145           |   THEN z := z * x
146           |   FI ;
147           |   x := x * x .
148           |

```

```

149 counterisnoteven counter is not even :
150           |   counter MOD 2 = 1 .
151           |
152           |ENDOP ** ;

```

153

```

154 min .....| INT PROC min (INT CONST first, second) :
155           |
156           |   IF first < second THEN first ELSE second FI
157           |
158           | ENDPROC min ;
159           |
    
```

```

160 max .....| INT PROC max (INT CONST first, second) :
161           |
162           |   IF first > second THEN first ELSE second FI
163           |
164           | ENDPROC max ;
165           |
166           |
167           |
168           | BOOL VAR conversion ok := TRUE ;
169           |
    
```

```

170 lastconversionok .....| BOOL PROC last conversion ok :
171           |   conversion ok
172           | ENDPROC last conversion ok ;
173           |
    
```

```

174 setconversion .....| PROC set conversion (BOOL CONST success) :
175           |   conversion ok := success
176           | ENDPROC set conversion ;
177           |
178           |
179           |
    
```

```

180           | (*****
181           | (*                                     *)
182           | (*                                     *)
183           | (*          RANDOM GENERATOR              *)
184           | (*                                     *)
185           | (*          x      := 4095 * x MOD (4095*4096+4093) *)
186           | (*          n+1      n                                     *)
187           | (*                                     *)
188           | (*          Periode: 2**24-4 > 16.0e6      *)
189           | (*                                     *)
190           | (*          Beachte: x = 4096 * x1 + x0, 0 <= x0,x1 < 4096 *)
191           | (*                                     *)
192           | (*****
193           |
194           |
195           | INT VAR high := 1, low := 0 ;
196           |
    
```

```

197 initializerandom .....| PROC initialize random (INT CONST start) :
198           |
199           |   low := start MOD 4096 ;
200           |   IF start < 0
201           |     THEN high := 256 + 16 + start DIV 4096 ;
202           |     IF low <> 0 THEN high DECR 1 FI
203           |     ELSE high := 256 + start DIV 4096
204           |   FI
    
```

```

205 |
206 |ENDPROC initialize random ;
207 |

208 random .....|INT PROC random (INT CONST lower bound, upper bound) :
209 |
210 |   compute new random value ;
211 |   normalize high ;
212 |   normalize low ;
213 |   map into interval .
214 |

215 computenewrandomvalue |compute new random value :
216 |   (* (high,low) := (low-high , 3*high-low) *)
217 |   high := low - high ;
218 |   low INCR low - 3 * high ;
219 |

220 normalizehigh |normalize high :
221 |   IF high < 0
222 |     THEN high INCR 4096 ; low DECR 3
223 |   FI .
224 |

225 normalizelow |normalize low :
226 |   (* high INCR low DIV 4096 ;
227 |     low := low MOD 4096
228 |   *)
229 |   IF low >= 4096 THEN low overflow
230 |   ELIF low < 0 THEN low underflow
231 |   FI .
232 |

233 lowoverflow |low overflow :
234 |   IF low >= 8192
235 |     THEN low DECR 8192 ; high INCR 2
236 |     ELSE low DECR 4096 ; high INCR 1 ; post normalization
237 |   FI .
238 |

239 postnormalization |post normalization :
240 |   (* IF (high,low) >= (4095,4093)
241 |     THEN (high,low) DECR (4095,4093)
242 |     FI
243 |   *)
244 |   IF high >= 4095
245 |     THEN IF low >= 4093 THEN high DECR 4095 ; low DECR 4093
246 |           ELIF high = 4096 THEN high := 0 ; low INCR 3
247 |           FI
248 |     FI .
249 |

250 lowunderflow |low underflow :
251 |   low INCR 4096 ; high DECR 1 .
252 |

253 mapintointerval |map into interval :
254 |   INT VAR number := high MOD 16 - 8 ;
255 |   number INCR 4095 * number + low ;
256 |   IF lower bound <= upper bound
257 |     THEN lower bound + number MOD (upper bound - lower bound + 1)

```

Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** integer

```
258 | ELSE upper bound + number MOD (lower bound - upper bound + 1)
259 | FI .
260 |
261 |ENDPROC random ;
262 |
263 |
264 |ENDPACKET integer ;
```

```

1
2 errorhandling *****|PACKET error handling DEFINES
3
4         enable stop ,
5         disable stop ,
6         is error ,
7         clear error ,
8         errormessage ,
9         error code ,
10        error line ,
11        put error ,
12        errorstop ,
13        stop :
14
15
16        LET cr lf          = "13"10" ,
17        line nr field     = 1 ,
18        error line field = 2 ,
19        error code field = 3 ,
20        syntax error code= 100 ,
21
22        error pre         = "7"13"10"5"FEHLER : " ;
23
24
25        TEXT VAR errortext := "" ;
26
27
28 enablestop .....|PROC enable stop :
29         EXTERNAL 75
30         ENDPROC enable stop ;
31
32 disablestop .....|PROC disable stop :
33         EXTERNAL 76
34         ENDPROC disable stop ;
35
36 seterrorstop .....|PROC set error stop (INT CONST code) :
37         EXTERNAL 77
38         ENDPROC set error stop ;
39
40 iserror .....|BOOL PROC is error :
41         EXTERNAL 78
42         ENDPROC is error ;
43
44 clearerror .....|PROC clear error :
45         EXTERNAL 79
46         ENDPROC clear error ;
47
48 selecterrormessage .....|PROC select error message :
49
50         SELECT error code OF

```

```

51          | CASE 1 : error text := "'halt' vom Terminal"
52          | CASE 2 : error text := "Stack-Ueberlauf"
53          | CASE 3 : error text := "Heap-Ueberlauf"
54          | CASE 4 : error text := "INT-Ueberlauf"
55          | CASE 5 : error text := "DIV durch 0"
56          | CASE 6 : error text := "REAL-Ueberlauf"
57          | CASE 7 : error text := "TEXT-Ueberlauf"
58          | CASE 8 : error text := "zu viele DATASPACEs"
59          | CASE 9 : error text := "Ueberlauf bei Subskription"
60          | CASE 10: error text := "Unterlauf bei Subskription"
61          | CASE 11: error text := "falscher DATASPACE-Zugriff"
62          | CASE 12: error text := "INT nicht initialisiert"
63          | CASE 13: error text := "REAL nicht initialisiert"
64          | CASE 14: error text := "TEXT nicht initialisiert"
65          | CASE 15: error text := "nicht implementiert"
66          | CASE 16: error text := "Block unlesbar"
67          | CASE 17: error text := "Codefehler"
68          | END SELECT
69
70          | ENDPROC select error message ;
71
72 errormessage ..... | TEXT PROC error message :
73
74          |   select error message ;
75          |   error text
76
77          | ENDPROC error message ;
78
79 errorcode ..... | INT PROC error code :
80
81          |   pcb (error code field)
82
83          | ENDPROC error code ;
84
85 errorline ..... | INT PROC error line :
86
87          |   IF is error
88          |     THEN pcb (error line field)
89          |     ELSE 0
90          |   FI
91
92          | ENDPROC error line ;
93
94 syntaxerror ..... | PROC syntax error (TEXT CONST message) :
95
96          |   INTERNAL 259 ;
97          |   errorstop (syntax error code, message) .
98
99          | ENDPROC syntax error ;
100

```

```

101 errorstop .....|PROC errorstop (TEXT CONST message) :
102                  |
103                  |   errorstop (0, message) ;
104                  |
105                  |ENDPROC errorstop ;
106                  |

107 errorstop .....|PROC errorstop (INT CONST code, TEXT CONST message) :
108                  |
109                  |   IF NOT is error
110                  |       THEN error text := message ;
111                  |           set error stop (code)
112                  |   FI
113                  |
114                  |ENDPROC errorstop ;
115                  |

116 puterror .....|PROC put error :
117                  |
118                  |   IF is error
119                  |       THEN select error message ;
120                  |           IF error text <> ""
121                  |               THEN put error message
122                  |           FI
123                  |   FI .
124                  |

125 puterrormessage |put error message :
126                  |   out (error pre) ;
127                  |   out (error text) ;
128                  |   IF error line > 0
129                  |       THEN out (" bei Zeile "); out (text (error line));
130                  |   FI ;
131                  |   out (cr lf) .
132                  |
133                  |ENDPROC put error ;
134                  |

135 stop .....|PROC stop :
136                  |
137                  |   errorstop ("stop")
138                  |
139                  |ENDPROC stop ;
140                  |
141                  |ENDPACKET error handling ;

```

```

1      | (* ----- VERSION 6    05.05.86 ----- *
2  real *****| PACKET real DEFINES    (* Autor: J.Liedtke *
3
4      |     text ,
5      |     int ,
6      |     real ,
7      |     round ,
8      |     floor ,
9      |     frac ,
10     |     decimal exponent ,
11     |     set exp ,
12     |     INCR ,
13     |     DECR ,
14     |     abs ,
15     |     ABS ,
16     |     sign ,
17     |     SIGN ,
18     |     MOD ,
19     |     min ,
20     |     max ,
21     |     max real ,
22     |     small real :
23
24     | LET mantissa length = 13 ,
25     |     digit zero index = 1 ,
26     |     digit nine index = 10 ;
27     | INT CONST
28     |     decimal point index := -1 ;
29
30     | TEXT VAR mantissa ;
31
32     | ROW 10 REAL VAR real digit ;
33
34     | INT VAR i ; REAL VAR d := 0.0 ;
35     | FOR i FROM 1 UPTO 10 REP
36     |     real digit (i) := d ;
37     |     d := d + 1.0
38     | PER ;
39
40     | maxreal .....| REAL PROC max real : 9.999999999999e126 ENDPROC max real ;
41
42     | smallreal .....| REAL PROC small real : 1.0e-12 ENDPROC small real ;
43
44     | sld .....| PROC sld (INT CONST in, REAL VAR real, INT VAR out) :
45     |     EXTERNAL 96
46     |     ENDPROC sld ;
47
48     | decimalexponent .....| INT PROC decimal exponent (REAL CONST mantissa) :
49     |     EXTERNAL 97
50     |     ENDPROC decimal exponent ;
51

```

```

52 setexp .....|PROC set exp (INT CONST exponent, REAL VAR number) :
53              |EXTERNAL 98
54              |ENDPROC set exp ;
55              |

56 tenpower .....|REAL PROC tenpower (INT CONST exponent) :
57              |REAL VAR result := 1.0 ;
58              |set exp (exponent, result) ;
59              |result
60              |ENDPROC tenpower ;
61              |

62 floor .....|REAL PROC floor (REAL CONST real) :
63              |EXTERNAL 99
64              |ENDPROC floor ;
65              |

66 round .....|REAL PROC round (REAL CONST real, INT CONST digits) :
67              |
68              |REAL VAR result := real ;
69              |IF (real <> 0.0) CAND (decimal exponent (real) + digits < mantissa
70              |length)
71              |THEN round result ;
72              |FI ;
73              |result .

74 roundresult .....|round result :
75              |set exp (decimal exponent (result) + digits, result) ;
76              |IF result >= 0.0
77              |THEN result := floor (result + 0.5)
78              |ELSE result := floor (result - 0.5)
79              |FI ;
80              |IF result <> 0.0
81              |THEN set exp (decimal exponent (result) - digits, result)
82              |FI .
83              |
84              |ENDPROC round ;
85              |
86              |TEXT VAR result ;
87              |

88 text .....|TEXT PROC text (REAL CONST real) :
89              |
90              |REAL VAR value := rounded to seven digits ;
91              |IF value = 0.0
92              |THEN "0.0"
93              |ELSE
94              |process sign ;
95              |get mantissa (value) ;
96              |INT CONST exponent := decimal exponent (value) ;
97              |get short mantissa ;
98              |IF exponent > 7 OR exponent < LENGTH short mantissa - 7
99              |THEN scientific notation
100             |ELSE short notation
101             |FI
102             |FI .
103             |

```

```

104 roundedtosevendigits |rounded to seven digits :
105 |round ( real * tenpower( -decimal exponent(real) ) , 6 )
106 |* tenpower ( decimal exponent(real) ) .
107 |
108 processsign |process sign :
109 |IF value < 0.0
110 |THEN result := "-" ;
111 |value := - value
112 |ELSE result := ""
113 |FI .
114 |
115 getshortmantissa |get short mantissa :
116 |INT VAR i := 7 ;
117 |WHILE (mantissa SUB i) = "0" REP
118 |i DECR 1
119 |UNTIL i=1 END REP ;
120 |TEXT CONST short mantissa := subtext (mantissa, 1, i) .
121 |
122 scientificnotation |scientific notation :
123 |result CAT (mantissa SUB 1) ;
124 |result CAT "." ;
125 |result CAT subtext (mantissa, 2, 7) ;
126 |result + "e" + text (exponent) .
127 |
128 shortnotation |short notation :
129 |IF exponent < 0
130 |THEN result + "0." + (-exponent - 1) * "0" + short mantissa
131 |ELSE result CAT subtext (short mantissa, 1, exponent+1) ;
132 |result CAT (exponent+1 - LENGTH short mantissa) * "0" ;
133 |result CAT "." ;
134 |result CAT subtext (short mantissa, exponent+2) ;
135 |IF LENGTH short mantissa < exponent + 2
136 |THEN result + "0"
137 |ELSE result
138 |FI
139 |FI .
140 |
141 |ENDPROC text ;
142 |
143 getmantissa ..... |PROC get mantissa (REAL CONST number) :
144 |
145 |REAL VAR real mantissa := number ;
146 |mantissa := "" ;
147 |INT VAR i , digit ;
148 |FOR i FROM 1 UPTO mantissa length REP
149 |sld (0, real mantissa, digit) ;
150 |mantissa CAT code (digit + 48)
151 |PER ;
152 |
153 |ENDPROC get mantissa ;
154 |
155 text ..... |TEXT PROC text (REAL CONST real, INT CONST length) :
156 |
157 |INT CONST mantissa length := min (length - 7, 13) ;

```

```

158      | IF mantissa length > 0
159      | THEN construct scientific notation
160      | ELSE result := length * "*"
161      | FI ;
162      | result .
163
164  constructscientificnot | construct scientific notation :
165      | REAL VAR value := rounded real ;
166      | IF value = 0.0
167      | THEN result := subtext (" 0.0", 1, length)
168      | ELSE process sign ;
169      | process mantissa ;
170      | process exponent
171      | FI .
172
173  roundedreal | rounded real :
174      | round (real * tenpower ( -decimal exponent (real) ), mantissa
175      | +
176      | length - 1)
177      | * tenpower (decimal exponent (real)) .
178
179  processsign | process sign :
180      | IF value < 0.0
181      | THEN result := "-"
182      | ELSE result := "+"
183      | FI .
184
185  processmantissa | process mantissa :
186      | get mantissa (value) ;
187      | result CAT (mantissa SUB 1) ;
188      | result CAT "." ;
189      | result CAT subtext (mantissa, 2, mantissa length) .
190
191  processexponent | process exponent :
192      | IF decimal exponent (value) >= 0
193      | THEN result CAT "e."
194      | ELSE result CAT "e-"
195      | FI ;
196      | result CAT text (ABS decimal exponent (value), 3) ;
197      | change all (result, " ", "0") .
198      | ENDPROC text ;
199
200  text ..... | TEXT PROC text (REAL CONST real, INT CONST length, fracs) :
201      | REAL VAR value := round (real, fracs) ;
202      | INT VAR exponent := decimal exponent (value) ;
203      | IF value = 0.0 THEN exponent := 0 FI ;
204      | INT VAR floors := exponent + 1 ,
205      | floor length := length - fracs - 1 ;
206      | IF value < 0.0 THEN floor length DECR 1 FI ;
207
208      | IF value too big
209      | THEN length * "*"
210      | ELSE transformed value
211      | FI .

```

```

212
213 transformedvalue transformed value :
214 process leading blanks and sign ;
215 get mantissa (value) ;
216 result CAT subtext (mantissa, 1, floors) ;
217 IF LENGTH mantissa < floors
218 THEN result CAT (floors - LENGTH mantissa) * "0"
219 FI ;
220 result CAT "." ;
221 IF exponent < 0
222 THEN result CAT (-floors) * "0" ;
223 result CAT subtext (mantissa, 1, length - LENGTH result)
224 ELSE result CAT subtext (mantissa, floors+1, floors + fracs)
225 FI ;
226 IF LENGTH result < length
227 THEN result CAT (length - LENGTH result) * "0"
228 FI ;
229 result .
230
231 processleadingblanksan process leading blanks and sign :
232 result := (floor length - max(floors,0)) * " " ;
233 IF value < 0.0
234 THEN result CAT "-";
235 value := - value
236 FI .
237
238 valuetoobig value too big :
239 floors > floor length .
240
241 ENDPROC text ;
242
243 real ..... REAL PROC real (TEXT CONST text) :
244 skip leading blanks ;
245 sign ;
246 mantissa part ;
247 exponent ;
248 result .
249
250
251 skipleadingblanks skip leading blanks :
252 INT VAR pos := 1 ;
253 skip blanks .
254
255 skipblanks skip blanks :
256 WHILE (text SUB pos) = " " REP
257 pos INCR 1
258 PER .
259
260 sign sign :
261 BOOL VAR negative ;
262 IF (text SUB pos) = "-"
263 THEN negative := TRUE ;
264 pos INCR 1
265 ELIF (text SUB pos) = "+"

```

```

266             THEN negative := FALSE ;
267             pos INCR 1
268         ELSE   negative := FALSE
269         FI .
270
271 mantissapart      mantissa part:
272                   REAL VAR value ;
273                   INT VAR exponent pos := 0 ;
274                   get first digit ;
275                   WHILE pos <= LENGTH text REP
276                   digit := code (text SUB pos) - 47 ;
277                   IF digit >= digit zero index AND digit <= digit nine index
278                   THEN value := value * 10.0 + real digit (digit) ;
279                   pos INCR 1
280                   ELIF digit = decimal point index AND exponent pos = 0
281                   THEN pos INCR 1 ;
282                   exponent pos := pos
283                   ELSE LEAVE mantissa part
284                   FI
285                   END REP .
286
287 getfirstdigit     get first digit :
288                   INT VAR digit := code (text SUB pos) - 47 ;
289                   IF digit = decimal point index
290                   THEN pos INCR 1 ;
291                   exponent pos := pos ;
292                   digit := code (text SUB pos) - 47
293                   FI ;
294                   IF digit >= digit zero index AND digit <= digit nine index
295                   THEN value := real digit (digit) ;
296                   pos INCR 1
297                   ELSE set conversion (FALSE) ;
298                   LEAVE real WITH 0.0
299                   FI .
300
301 exponent          exponent :
302                   INT VAR exp ;
303                   IF exponent pos > 0
304                   THEN exp := exponent pos - pos
305                   ELSE exp := 0
306                   FI ;
307                   IF (text SUB pos) = "e"
308                   THEN exp INCR int (subtext(text,pos+1))
309                   ELSE no more nonblank chars permitted
310                   FI .
311
312 nomorenoblankcharsper no more nonblank chars permitted :
313                   skip blanks ;
314                   IF pos > LENGTH text
315                   THEN set conversion (TRUE)
316                   ELSE set conversion (FALSE)
317                   FI .
318
319 result            result :
320                   value := value * tenpower (exp) ;
321                   IF negative
322                   THEN - value

```

```

323         ELSE value
324         FI .
325
326     ENDPROC real ;
327
328
329 abs .....|REAL PROC abs (REAL CONST value) :
330
331         IF value >= 0.0
332         THEN value
333         ELSE -value
334         FI
335
336     ENDPROC abs ;
337
338 ABS .....|REAL OP ABS (REAL CONST value) :
339
340         abs (value)
341
342     ENDP ABS ;
343
344 sign .....|INT PROC sign (REAL CONST value) :
345
346         IF value < 0.0 THEN -1
347         ELIF value = 0.0 THEN 0
348         ELSE 1
349         FI
350
351     ENDPROC sign ;
352
353 SIGN .....|INT OP SIGN (REAL CONST value) :
354
355         sign (value)
356
357     ENDP SIGN ;
358
359 MOD .....|REAL OP MOD (REAL CONST left, right) :
360
361         REAL VAR result := left - floor (left/right) * right ;
362         IF result < 0.0
363         THEN result + abs (right)
364         ELSE result
365         FI
366
367     ENDP MOD ;
368
369 frac .....|REAL PROC frac (REAL CONST value) :
370
371         value - floor (value)
372

```

```

373                                     |ENDPROC frac ;
374                                     |
375 max .....|REAL PROC max (REAL CONST a, b) :
376                                     |
377                                     |   IF a > b THEN a ELSE b FI
378                                     |
379                                     |ENDPROC max ;
380                                     |
381 min .....|REAL PROC min (REAL CONST a, b) :
382                                     |
383                                     |   IF a < b THEN a ELSE b FI
384                                     |
385                                     |ENDPROC min ;
386                                     |
387 INCR .....|OP INCR (REAL VAR dest, REAL CONST increment) :
388                                     |
389                                     |   dest := dest + increment
390                                     |
391                                     |ENDOP INCR ;
392                                     |
393 DECR .....|OP DECR (REAL VAR dest, REAL CONST decrement) :
394                                     |
395                                     |   dest := dest - decrement
396                                     |
397                                     |ENDOP DECR ;
398                                     |
399 int .....|INT PROC int (REAL CONST value) :
400                                     |
401                                     |   IF value = minint value
402                                     |       THEN minint
403                                     |       ELSE compute int result ;
404                                     |           IF value < 0.0
405                                     |               THEN - result
406                                     |               ELSE result
407                                     |           FI
408                                     |   FI .
409                                     |
410 computeintresult compute int result :
411                                     |   INT VAR result := 0, digit ,i ;
412                                     |   REAL VAR mantissa := value ;
413                                     |
414                                     |   FOR i FROM 0 UPTO decimal exponent (value) REP
415                                     |       sld (0, mantissa, digit) ;
416                                     |       result := result * 10 + digit
417                                     |   PER .
418                                     |
419 minintvalue minint value : - 32768.0 .

```

Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** real

```
420 minint |minint : - 32767 - 1 .
421
422 |ENDPROC int ;
423
424 real .....|REAL PROC real (INT CONST value) :
425
426 | IF value < 0
427 | THEN - real (-value)
428 | ELIF value < 10
429 | THEN real digit (value+1)
430 | ELSE split value into head and last digit ;
431 | real (head) * 10.0 + real digit (last digit+1)
432 | FI .
433
434 splitvalueintoheadandl |split value into head and last digit :
435 | INT CONST
436 | head := value DIV 10 ,
437 | last digit := value - head * 10 .
438
439 |ENDPROC real ;
440
441 |ENDPACKET real ;
```

```

1  datehandling *****|PACKET date handling DEFINES date, time,      (* Autor: H.
+                               |Indenbirken *)
2                               |
+                               |time of day,      (* Stand:
3                               |02.06.1986 (wk)*)
4                               |month, day , year ,
5                               |hour ,
6                               |minute,
7                               |second :
8                               |
9                               |LET middle yearlength = 31557380.0,
10                              |weeklength = 604800.0,
11                              |daylength = 86400.0,
12                              |hours = 3600.0,
13                              |minutes = 60.0,
14                              |seconds = 1.0;
15                              |(* Tage bis zum Jahr 01.01.1900: 693970.25 5.995903e10 Sekunden
+                               |*)
16                              |(* Dieser Tag ist ein Montag
+                               |*)
17                              |
18                              |REAL VAR begin of today := 0.0 , end of today := 0.0 ;
19                              |
20                              |TEXT VAR today , result ;
21                              |
22                              |
23                              |ROW 12 REAL CONST previous days :: ROW 12 REAL : (0.0, 2678400.0,
+                               |5097600.0,
24                              |7776000.0, 10368000.0,
+                               |13046400.0,
25                              |15638400.0, 18316800.0,
+                               |20995200.0,
26                              |23587200.0, 26265600.0,
+                               |28857600.0);
27                              |
28  day .....|REAL PROC day: day length END PROC day;
29  hour .....|REAL PROC hour: hours END PROC hour;
30  minute .....|REAL PROC minute: minutes END PROC minute;
31  second .....|REAL PROC second: seconds END PROC second;
32
33  date .....|TEXT PROC date :
34
35          |IF clock (1) < begin of today OR end of today <= clock (1)
36          |THEN begin of today := clock (1) ;
37          |end of today := floor (begin of
+          |today/daylength)*daylength+daylength;
38          |today := date (begin of today)
39          |FI ;
40          |today
41
42          |ENDPROC date ;
43

```

```

44 date .....|TEXT PROC date (REAL CONST datum):
45             |INT VAR year :: int (datum/middle yearlength),
46             |day :: int (((datum - datum MOD daylength) MOD middle
+             |yearlength) / daylength) + 1;
47
48             |correct kalendary day;
49
50             |calculate month and correct day;
51             |result := daytext;
52             |result CAT monthtext;
53             |result CAT yeartext;
54             |change all (result, " ", "0") ;
55             |result .
56
57 correctkalendaryday |correct kalendary day:
58             |IF day >= 60 AND NOT leapyear
59             |THEN day INCR 1 FI .
60
61 leapyear          |leapyear:
62             |IF year MOD 100 = 0
63             |THEN year MOD 400 = 0
64             |ELSE year MOD 4 = 0
65             |FI.
66
67 calculatemonthandcorre |calculate month and correct day:
68             |INT VAR month;
69             |IF day > 182
70             |THEN IF day > 274
71             |THEN IF day > 305
72             |THEN IF day > 335
73             |THEN month := 12;
74             |day DECR 335
75             |ELSE month := 11;
76             |day DECR 305
77             |FI
78             |ELSE month := 10;
79             |day DECR 274
80             |FI
81             |ELSE IF day > 213
82             |THEN IF day > 244
83             |THEN month := 9;
84             |day DECR 244
85             |ELSE month := 8;
86             |day DECR 213
87             |FI
88             |ELSE month := 7;
89             |day DECR 182
90             |FI
91             |FI
92             |ELSE IF day > 91
93             |THEN IF day > 121
94             |THEN IF day > 152
95             |THEN month := 6;
96             |day DECR 152
97             |ELSE month := 5;
98             |day DECR 121
99             |FI
100            |ELSE month := 4;
101            |day DECR 91

```

```

102          FI
103          ELSE IF day > 31
104          THEN IF day > 60
105              THEN month := 3;
106                  day DECR 60
107              ELSE month := 2;
108                  day DECR 31
109          FI
110          ELSE month := 1 FI
111      FI
112      FI .
113
114      daytext      daytext :
115                    text (day, 2) + "."
116
117      monthtext    monthtext :
118                    text (month,2) + "."
119
120      yeartext     yeartext:
121                    IF 1900 <= year AND year < 2000
122                        THEN text (year - 1900, 2)
123                        ELSE text (year, 4)
124                    FI .
125
126      END PROC date;
127

```

```

128      day ..... TEXT PROC day (REAL CONST datum):
129                    SELECT int ((datum MOD weeklength)/daylength) OF
130                    CASE 1: "Donnerstag"
131                    CASE 2: "Freitag"
132                    CASE 3: "Samstag"
133                    CASE 4: "Sonntag"
134                    CASE 5: "Montag"
135                    CASE 6: "Dienstag"
136                    OTHERWISE "Mittwoch" ENDSELECT .
137      END PROC day;
138

```

```

139      month ..... TEXT PROC month (REAL CONST datum):
140                    SELECT int (subtext (date (datum), 4, 5)) OF
141                    CASE 1: "Januar"
142                    CASE 2: "Februar"
143                    CASE 3: "März"
144                    CASE 4: "April"
145                    CASE 5: "Mai"
146                    CASE 6: "Juni"
147                    CASE 7: "Juli"
148                    CASE 8: "August"
149                    CASE 9: "September"
150                    CASE 10: "Oktober"
151                    CASE 11: "November"
152                    OTHERWISE "Dezember" ENDSELECT .
153      END PROC month;
154
155

```

```

156 year .....|TEXT PROC year (REAL CONST datum) :
157           |
158           |   TEXT VAR buffer := subtext (date (datum), 7) ;
159           |   IF LENGTH buffer = 2
160           |     THEN "19" + buffer
161           |     ELSE buffer
162           |   FI .
163           |
164           |ENDPROC year ;
165           |

166 timeofday .....|TEXT PROC time of day :
167           |   time of day (clock (1))
168           |ENDPROC time of day ;
169           |

170 timeofday .....|TEXT PROC time of day (REAL CONST value) :
171           |   subtext (time (value MOD daylength), 1, 5)
172           |ENDPROC time of day ;
173           |

174 time .....|TEXT PROC time (REAL CONST value) :
175           |   time (value,10)
176           |ENDPROC time ;
177           |

178 time .....|TEXT PROC time (REAL CONST value, INT CONST length) :
179           |   result := "" ;
180           |   IF length > 7
181           |     THEN result CAT hour ;
182           |           result CAT ":"
183           |   FI ;
184           |   result CAT minute ;
185           |   result CAT ":" ;
186           |   result CAT rest ;
187           |   change all (result, " ", "0") ;
188           |   result .
189           |

190   hour           |hour :
191           |   text (int (value/hours), length-8) .
192           |

193   minute         |minute :
194           |   text (int (value/minutes MOD 60.0), 2) .
195           |

196   rest           |rest :
197           |   text (value MOD minutes, 4, 1) .
198           |
199           |END PROC time ;
200           |

201 date .....|REAL PROC date (TEXT CONST datum) :
202           |   split and check datum;
203           |   real (day no)*daylength +
204           |   previous days [month no] + calendary day +

```

```

205      floor (real (year no)*middleyearlength / daylength)=daylength .
206
207      splitandcheckdatum  split and check datum:
208      INT CONST day no :: first no;
209      IF NOT last conversion ok
210      THEN errorstop ("inkorrekte Datumsangabe (Tag) : " + datum) FI;
211
212      INT CONST month no :: second no;
213      IF NOT last conversion ok OR month no < 1 OR month no > 12
214      THEN errorstop ("inkorrekte Datumsangabe (Monat) : " + datum) FI
215
216      INT CONST year no :: third no + century;
217      IF NOT last conversion ok
218      THEN errorstop ("inkorrekte Datumsangabe (Jahr) : " + datum) FI;
219
220      IF day no < 1 OR day no > size of month
221      THEN errorstop ("inkorrekte Datumsangabe (Tag) : " + datum) FI
222
223      century              century:
224      IF (length (datum) - second pos) <= 2
225      THEN 1900
226      ELSE 0 FI .
227
228      sizeofmonth         size of month:
229      SELECT month no OF
230      CASE 1, 3, 5, 7, 8, 10, 12: 31
231      CASE 4, 6, 9, 11: 30
232      OTHERWISE february size ENDSELECT .
233
234      februarysize       february size:
235      IF leapyear
236      THEN 29
237      ELSE 28 FI .
238
239      calendaryday       calendary day:
240      IF month no > 2 AND leapyear
241      THEN daylength
242      ELSE 0.0 FI .
243
244      leapyear           leapyear:
245      year no MOD 4 = 0 AND year no MOD 400 <> 0 .
246
247      firstno           first no:
248      INT CONST first pos :: pos (datum, ".");
249      int (subtext (datum, 1, first pos-1)) .
250
251      secondno          second no:
252      INT CONST second pos :: pos (datum, ".", first pos+1);
253      int (subtext (datum, first pos + 1, second pos-1)) .
254
255      thirdno           third no:
256      int (subtext (datum, second pos + 1)) .
257

```

```

258                                     |END PROC date;
259                                     |
260 time .....|REAL PROC time (TEXT CONST time) :
261             | split and check time;
262             | hour - min + sec .
263             |
264 splitandchecktime |split and check time:
265             | REAL CONST hour :: hour no * hours;
266             | IF NOT last conversion ok
267             | THEN errorstop ("inkorrekte Datumsangabe (Stunde) : " + time) FI;
268             |
269             | REAL CONST min :: min no * minutes;
270             | IF NOT last conversion ok
271             | THEN errorstop ("inkorrekte Datumsangabe (Minute) : " + time) FI;
272             |
273             | REAL CONST sec :: sec no;
274             | IF NOT last conversion ok
275             | THEN errorstop ("inkorrekte Datumsangabe (Sekunde) : " + time) FI;
276             |
277             | set conversion (hour ok AND min ok AND sec ok) .
278             |
279 hourno      |hour no:
280             | INT CONST hour pos :: pos (time, ":");
281             | real (subtext (time, 1, hour pos-1)) .
282             |
283 minno       |min no:
284             | INT VAR min pos :: pos (time, ":", hour pos+1);
285             | IF min pos = 0
286             | THEN real (subtext (time, hour pos + 1, LENGTH time))
287             | ELSE real (subtext (time, hour pos + 1, min pos-1))
288             | FI .
289             |
290 secno       |sec no:
291             | IF min pos = 0
292             | THEN 0.0
293             | ELSE real (subtext (time, min pos + 1))
294             | FI .
295             |
296 hourok      |hour ok: 0.0 <= hour AND hour < daylength .
297 minok       |min ok: 0.0 <= min AND min < hours .
298 secok       |sec ok: 0.0 <= sec AND sec < minutes .
299             |END PROC time;
300             |
301             |END PACKET datehandling

```

```

1
2 commanddialogue *****|PACKET command dialogue DEFINES                (* Autor: J.Liedtke *)
3                                                                    (* Stand: 25.11.83 *)
4                                                                    command dialogue ,
5                                                                    say ,
6                                                                    yes ,
7                                                                    no ,
8                                                                    param position ,
9                                                                    last param ,
10                                                                    std ,
11                                                                    QUIET ,
12                                                                    quiet :
13
14
15                                                                    LET up      = ""3"" ,
16                                                                    right     = ""2"" ,
17                                                                    cr lf    = ""13""10"" ,
18                                                                    param pre = "" "" ,
19                                                                    param post = """"13""10"" ;
20
21
22                                                                    TEXT VAR std param := "" ;
23
24                                                                    BOOL VAR dialogue flag := TRUE ;
25
26                                                                    INT VAR param x := 0 ;
27
28
29                                                                    TYPE QUIET = INT ;
30
31 quiet .....|QUIET PROC quiet :
32                                                                    QUIET:(0)
33                                                                    ENDPROC quiet ;
34
35
36 commanddialogue .....|BOOL PROC command dialogue :
37                                                                    dialogue flag
38                                                                    ENDPROC command dialogue ;
39
40 commanddialogue .....|PROC command dialogue (BOOL CONST status) :
41                                                                    dialogue flag := status
42                                                                    ENDPROC command dialogue ;
43
44
45 yes .....|BOOL PROC yes (TEXT CONST question) :
46
47                                                                    IF dialogue flag
48                                                                    THEN ask question
49                                                                    ELSE TRUE
50                                                                    FI .
51

```

```

52 askquestion ask question :
53 out (question) ;
54 skip previous input chars ;
55 out (" (j/n) ? ") ;
56 get answer ;
57 IF correct answer
58 THEN out (answer) ;
59 out (cr lf) ;
60 positive answer
61 ELSE out ("7") ;
62 LENGTH question + 9 TIMEOUT "8" ;
63 yes (question)
64 FI .
65
66 getanswer get answer :
67 TEXT VAR answer ;
68 inchar (answer) .
69
70 correctanswer correct answer :
71 pos ("jnyJNY", answer) > 0 .
72
73 positiveanswer positive answer :
74 pos ("jyJY", answer) > 0 .
75
76 skippreviousinputchars skip previous input chars :
77 REP UNTIL incharety = "" PER .
78
79 ENDPROC yes ;
80
81 no ..... BOOL PROC no (TEXT CONST question) :
82
83 NOT yes (question)
84
85 ENDPROC no ;
86
87 say ..... PROC say (TEXT CONST message) :
88
89 IF dialogue flag
90 THEN out (message)
91 FI
92
93 ENDPROC say ;
94
95 paramposition ..... PROC param position (INT CONST x) :
96
97 param x := x
98
99 ENDPROC param position ;
100

```

```
101 lastparam .....|TEXT PROC last param :
102                |
103                |   IF param x > 0 AND online
104                |       THEN out (up) ;
105                |           param x TIMESOUT right ;
106                |           out (param pre) ;
107                |           out (std param) ;
108                |           out (param post)
109                |   FI ;
110                |   std param .
111                |
112                |ENDPROC last param ;
113                |

114 lastparam .....|PROC last param (TEXT CONST new) :
115                |   std param := new
116                |ENDPROC last param ;
117                |

118 std .....|TEXT PROC std :
119                |   std param
120                |ENDPROC std ;
121                |
122                |ENDPACKET command dialogue ;
```

```

1      |(* ----- VERSION 2      06.03.86 ----- *)
2  thesaurushandling *****|PACKET thesaurus handling      (* Autor: J.Liedtke *)
3
4      |      DEFINES      THESAURUS ,
5      |                  := ,
6      |                  empty thesaurus ,
7      |                  insert,      (* fuegt ein Element ein *)
8      |                  delete,      (* loescht ein Element falls
+      |                  vorhanden*)
9      |                  rename,      (* aendert ein Element falls
+      |                  vorhanden*)
10     |                  CONTAINS ,      (* stellt fest, ob enthalten *)
11     |                  link ,      (* index in thesaurus *)
12     |                  name ,      (* name of entry *)
13     |                  get ,      (* get next entry (" is eof)*)
14     |                  highest entry : (* highest valid index of thes*)
15
16     |
17     |TYPE THESAURUS = TEXT ;
18
19     |LET thesaurus size = 200 ,
20     |     nil = 0 ,
21     |     niltext = "" ,
22     |     max name length = 80 ,
23
24     |     begin entry char = ""0"" ,
25     |     end entry char = ""1"" ,
26
27     |     nil entry = ""0""1"" ,
28     |     nil name = "" ,
29
30     |     quote = """" ;
31
32     |TEXT VAR entry ;
33     |INT VAR cache index := 0 ,
34     |     cache pos ;
35
36
37  access .....|PROC access (THESAURUS CONST thesaurus, TEXT CONST name) :
38
39     |construct entry ;
40     |IF NOT cache identifies entry
41     |THEN search through thesaurus list
42     |FI ;
43     |IF entry found
44     |THEN cache index := code (list SUB (cache pos - 1))
45     |ELSE cache index := 0
46     |FI .
47
48  constructentry|construct entry :
49     |entry := begin entry char ;
50     |entry CAT name ;
51     |decode invalid chars (entry, 2) ;
52     |entry CAT end entry char .
53
54  searchthroughthesaurus|search through thesaurus list :
55     |cache pos := pos (list, entry) .
56

```

```

57  cacheidentifiesentry |cache identifies entry :
58                        |cache pos <> 0 AND
59                        |pos (list, entry, cache pos, cache pos + LENGTH entry) = cache pos
+
60                        |
61  entryfound           |entry found : cache pos > 0 .
62
63  list                 |list : CONCR (thesaurus) .
64
65                      |ENDPROC access ;
66

67  access .....        |PROC access (THESAURUS CONST thesaurus, INT CONST index) :
68
69                      |IF cache identifies index
70                      |THEN cache index := index ;
71                      |construct entry
72                      |ELSE cache pos := pos (list, code (index) + begin entry char) ;
73                      |IF entry found
74                      |THEN cache pos INCR 1 ;
75                      |cache index := index ;
76                      |construct entry
77                      |ELSE cache index := 0 ;
78                      |entry := niltext
79                      |FI
80                      |FI .
81

82  constructentry      |construct entry :
83                      |entry := subtext (list, cache pos, pos (list, end entry char,
+                      |cache pos)) .
84

85  cacheidentifiesindex |cache identifies index :
86                      |subtext (list, cache pos-1, cache pos) = code (index) + begin
+                      |entry char .
87

88  entryfound           |entry found : cache pos > 0 .
89

90  list                 |list : CONCR (thesaurus) .
91
92                      |ENDPROC access ;
93
94
95

96  emptythesaurus .....|THESAURUS PROC empty thesaurus :
97
98                      |THESAURUS : ("1")
99
100                     |ENDPROC empty thesaurus ;
101
102

```

```

103 := ..... OP := (THESAURUS VAR dest, THESAURUS CONST source) :
104
105         CONCR (dest) := CONCR (source) .
106
107         ENDOP := ;
108
109         TEXT VAR insert name ;
110
111 insert ..... PROC insert (THESAURUS VAR thesaurus, TEXT CONST name, INT VAR
+         index) :
112
113         insert name := name ;
114         decode invalid chars (insert name, 1) ;
115         IF insert name = "" OR LENGTH insert name > max name length
116           THEN index := nil ; errorstop ("Name unzulassig")
117           ELSE insert element
118         FI .
119
120 insertelement insert element :
121         search free entry ;
122         IF entry found
123           THEN insert into directory
124           ELSE add entry to directory if possible
125         FI .
126
127 searchfreeentry search free entry :
128         access (thesaurus, nil name) .
129
130 insertintodirectory insert into directory :
131         change (list, cache pos + 1, cache pos, insert name) ;
132         index := cache index .
133
134 addentrytodirectoryifp add entry to directory if possible :
135         INT CONST next free index := code (list SUB LENGTH list) ;
136         IF next free index <= thesaurus size
137           THEN add entry to directory
138           ELSE directory overflow
139         FI .
140
141 addentrytodirectory add entry to directory :
142         list CAT begin entry char ;
143         cache pos := LENGTH list ;
144         cache index := next free index ;
145         list CAT insert name ;
146         list CAT end entry char + code (next free index + 1) ;
147         index := cache index .
148
149 directoryoverflow directory overflow :
150         index := nil .
151
152 entryfound entry found : cache index > @ .
153

```

```

154 list |list : CONCR (thesaurus) .
155 |
156 |ENDPROC insert ;
157 |

158 decodeinvalidchars .....|PROC decode invalid chars (TEXT VAR name, INT CONST start pos) :
159 |
160 | INT VAR invalid char pos := pos (name, ""0"", ""31"", start pos) ;
161 | WHILE invalid char pos > 0 REP
162 | change (name, invalid char pos, invalid char pos, decoded char) ;
163 | invalid char pos := pos (name, ""0"", ""31"", invalid char pos)
164 | PER .
165 |

166 decodedchar |decoded char : quote + text(code(name SUB invalid char pos)) +
+ | quote.
167 |
168 |ENDPROC decode invalid chars ;
169 |

170 insert .....|PROC insert (THESAURUS VAR thesaurus, TEXT CONST name) :
171 |
172 | INT VAR index ;
173 | insert (thesaurus, name, index) ;
174 | IF index = nil AND NOT is error
175 | THEN errorstop ("THESAURUS-Ueberlauf")
176 | FI .
177 |
178 |ENDPROC insert ;
179 |

180 delete .....|PROC delete (THESAURUS VAR thesaurus, TEXT CONST name, INT VAR
+ | index) :
181 |
182 | access (thesaurus, name) ;
183 | index := cache index ;
184 | delete (thesaurus, index) .
185 |
186 |ENDPROC delete ;
187 |

188 delete .....|PROC delete (THESAURUS VAR thesaurus, INT CONST index) :
189 |
190 | access (thesaurus, index) ;
191 | IF entry found
192 | THEN delete entry
193 | FI .
194 |

195 deleteentry |delete entry :
196 | IF is last entry of thesaurus
197 | THEN cut off as much as possible
198 | ELSE set to nil entry
199 | FI .
200 |

```

```

201 settonilentry |set to nil entry :
202 |   change (list, cache pos, cache pos + LENGTH entry - 1, nil entry)
+
203 |
204 cutoffasmuchaspossible |cut off as much as possible :
205 |   WHILE predecessor is also nil entry REP
206 |     set cache to this entry
207 |   PER ;
208 |   list := subtext (list, 1, cache pos - 1) ;
209 |   erase cache .
210 |
211 predecessorisalsonilen |predecessor is also nil entry :
212 |   subtext (list, cache pos - 3, cache pos - 2) = nil entry .
213 |
214 setcachetothisentry |set cache to this entry :
215 |   cache pos DECR 3 .
216 |
217 erasecache |erase cache :
218 |   cache pos := 0 ;
219 |   cache index := 0 .
220 |
221 islastentryofthesaurus |is last entry of thesaurus :
222 |   pos (list, end entry char, cache pos) = LENGTH list - 1 .
223 |
224 list |list :   CONCR (thesaurus) .
225 |
226 entryfound |entry found :   cache index > nil .
227 |
228 |ENDPROC delete ;
229 |
230 |
231 CONTAINS ..... |BOOL OP CONTAINS (THESAURUS CONST thesaurus, TEXT CONST name) :
232 |
233 |   IF name = niltext OR LENGTH name > max name length
234 |     THEN FALSE
235 |     ELSE access (thesaurus, name) ; entry found
236 |   FI .
237 |
238 entryfound |entry found :   cache index > nil .
239 |
240 |ENDOP CONTAINS ;
241 |
242 rename ..... |PROC rename (THESAURUS VAR thesaurus, TEXT CONST old, new) :
243 |
244 |   rename (thesaurus, link (thesaurus, old), new)
245 |
246 |ENDPROC rename ;
247 |

```

```

248 rename .....|PROC rename (THESAURUS VAR thesaurus, INT CONST index, TEXT CONST
+              |new) :
249              |
250              |insert name := new ;
251              |decode invalid chars (insert name, 1) ;
252              |IF insert name = "" OR LENGTH insert name > max name length
253              |THEN errorstop ("Name unzuessaessig")
254              |ELSE change to new name
255              |FI .
256
257 changetonewname |change to new name :
258              |access (thesaurus, index) ;
259              |IF cache index <> 0 AND entry <> ""
260              |THEN change (list, cache pos + 1, cache pos + LENGTH entry - 2,
+              |insert name)
261              |FI .
262
263 list            |list : CONCR (thesaurus) .
264
265                |ENDPROC rename ;
266
267 link .....|INT PROC link (THESAURUS CONST thesaurus, TEXT CONST name) :
268
269              |access (thesaurus, name) ;
270              |cache index .
271
272              |ENDPROC link ;
273
274 name .....|TEXT PROC name (THESAURUS CONST thesaurus, INT CONST index) :
275
276              |access (thesaurus, index) ;
277              |subtext (entry, 2, LENGTH entry - 1) .
278
279              |ENDPROC name ;
280
281 get .....|PROC get (THESAURUS CONST thesaurus, TEXT VAR name, INT VAR index) :
282
283              |identify index ;
284              |REP
285              |to next entry
286              |UNTIL end of list COR valid entry found PER .
287
288 identifyindex  |identify index :
289              |IF index = 0
290              |THEN cache index := 0 ;
291              |cache pos := 1
292              |ELSE access (thesaurus, index)
293              |FI .
294
295 tonextentry    |to next entry :
296              |cache pos := pos (list, begin entry char, cache pos + 1) ;
297              |IF cache pos > 0

```

```

298          THEN get entry
299          ELSE get nil entry
300          FI .
301
302  getentry      get entry :
303                cache index INCR 1 ;
304                index := cache index ;
305                name := subtext (list, cache pos + 1, end entry pos - 1) .
306
307  getnilentry   get nil entry :
308                cache index := 0 ;
309                cache pos := 0 ;
310                index := 0 ;
311                name := "" .
312
313  endentrypos   end entry pos :   pos (list, end entry char, cache pos) .
314
315  endoflist     end of list :     index = 0 .
316
317  validentryfound valid entry found : name <> "" .
318
319  list          list :           CONCR (thesaurus) .
320
321              ENDPROC get ;
322
323  highestentry ..... INT PROC highest entry (THESAURUS CONST thesaurus) :
324                    +          (*840813*)
325                    code (list SUB LENGTH list) - 1 .
326
327  list          list :           CONCR (thesaurus) .
328
329              ENDPROC highest entry ;
330
331              ENDPACKET thesaurus handling ;

```

```

1      | (* ----- VERSION 2      24.02.86 -----)
2  localmanager *****| PACKET local manager      (* Autor: J.Liedtke
3
4      |
5      |   DEFINES
6      |       create,      (* neue lokale Datei einrichten *)
7      |       new,         (* 'create' und Datei liefern *)
8      |       old,        (* bestehende Datei liefern *)
9      |       forget,     (* lokale Datei loeschen *)
10     |       exists,     (* existiert Datei (lokal) ? *)
11     |       status,     (* setzt und liefert Status *)
12     |       rename,     (* Umbenennung *)
13     |       copy ,      (* Datenraum in Datei kopieren *)
14     |       enter password,(* Passwort einfuehren *)
15     |       write password ,
16     |       read password ,
17     |       write permission ,
18     |       read permission ,
19     |       begin list ,
20     |       get list entry ,
21     |       all :
22
23
24     | LET size = 200 ,
25     |       nil = 0 ;
26
27     | INT VAR index ;
28
29     | TEXT VAR system write password := "",
30     |       system read password := "",
31     |       actual password ;
32
33     | INITFLAG VAR this packet := FALSE ;
34
35     | DATASPACE VAR password space ;
36
37     | BOUND ROW size STRUCT (TEXT write, read) VAR passwords ;
38
39
40     | THESAURUS VAR dir := empty thesaurus ;
41
42     | ROW size STRUCT (DATASPACE ds,
43     |               BOOL protected,
44     |               TEXT status) VAR crowd ;
45
46
47  initializeifnecessary ....| PROC initialize if necessary :
48
49     | IF NOT initialized (this packet)
50     |     THEN system write password := "" ;
51     |     system read password := "" ;
52     |     dir := empty thesaurus ;
53     |     password space := nilspace ;
54     |     passwords := password space
55     |
56     | FI
57
58     | ENDPROC initialize if necessary ;
59
60

```

```

61 create .....|PROC create (TEXT CONST name) :
62
63             |IF exists (name )
64             |   THEN error (name, "existiert bereits") ;
65             |   index := nil
66             |   ELSE insert and initialize entry
67             |FI .
68
69 insertandinitializeent |insert and initialize entry :
70             |  disable stop ;
71             |  insert (dir, name, index) ;
72             |  IF index <> nil
73             |    THEN crowd (index).ds := nilspace ;
74             |    IF is error
75             |      THEN delete (dir, name, index) ;
76             |      LEAVE create
77             |    FI ;
78             |    status (name, "") ;
79             |    crowd (index).protected := FALSE
80             |  ELIF NOT is error
81             |    THEN errorstop ("zu viele Dateien")
82             |  FI .
83
84             |ENDPROC create ;
85
86 new .....|DATASPACE PROC new (TEXT CONST name) :
87
88             |  create (name) ;
89             |  IF index <> nil
90             |    THEN crowd (index).ds
91             |    ELSE nilspace
92             |  FI
93
94             |ENDPROC new ;
95
96 old .....|DATASPACE PROC old (TEXT CONST name) :
97
98             |  initialize if necessary ;
99             |  index := link (dir, name) ;
100            |  IF index = 0
101            |    THEN error (name, "gibt es nicht") ;
102            |    nilspace
103            |    ELSE space
104            |  FI .
105
106            |space : crowd (index).ds .
107
108            |ENDPROC old ;
109
110 old .....|DATASPACE PROC old (TEXT CONST name, INT CONST expected type) :
111
112            |  initialize if necessary ;
113            |  index := link (dir, name) ;
114            |  IF index = 0

```

```

115 |         THEN error (name, "gibt es nicht") ;
116 |         nilspace
117 |     ELIF type (space) <> expected type
118 |         THEN errorstop ("Datenraum hat falschen Typ") ;
119 |         nilspace
120 |     ELSE space
121 |     FI .
122 |
123 | space |space : crowd (index).ds .
124 |
125 |     ENDPROC old ;
126 |
127 | exists .....|BOOL PROC exists (TEXT CONST name) :
128 |
129 |         initialize if necessary ;
130 |         dir CONTAINS name
131 |
132 |     ENDPROC exists ;
133 |
134 | forget .....|PROC forget (TEXT CONST name) :
135 |
136 |         initialize if necessary ;
137 |         say (""") ;
138 |         say (name) ;
139 |         IF NOT exists (name) THEN say ("" existiert nicht")
140 |         ELIF yes ("" loeschen") THEN forget (name, quiet)
141 |         FI .
142 |
143 |     ENDPROC forget ;
144 |
145 | forget .....|PROC forget (TEXT CONST name, QUIET CONST q) :
146 |
147 |         initialize if necessary ;
148 |         disable stop ;
149 |         delete (dir, name, index) ;
150 |         IF index <> nil
151 |             THEN forget ( crowd (index).ds ) ;
152 |                 crowd (index).status := ""
153 |         FI .
154 |
155 |     ENDPROC forget ;
156 |
157 | forget .....|PROC forget :
158 |
159 |         BOOL VAR status := command dialogue ;
160 |         command dialogue (TRUE) ;
161 |         forget (last param) ;
162 |         command dialogue (status)
163 |
164 |     ENDPROC forget ;
165 |

```

```

166 status .....|PROC status (TEXT CONST name, status text) :
167                |
168                |   initialize if necessary ;
169                |   INT VAR index := link (dir, name) ;
170                |   IF index > 0
171                |       THEN crowd (index).status := date + " " + text (status text, 4)
172                |   FI
173                |
174                |ENDPROC status ;
175                |

```

```

176 status .....|TEXT PROC status (TEXT CONST name) :
177                |
178                |   initialize if necessary ;
179                |   INT VAR index := link (dir, name) ;
180                |   IF index > 0
181                |       THEN crowd (index).status
182                |           ELSE ""
183                |   FI
184                |
185                |ENDPROC status ;
186                |

```

```

187 status .....|PROC status (INT CONST pos, TEXT CONST status pattern) :
188                |
189                |   initialize if necessary ;
190                |   INT VAR index := 0 ;
191                |   WHILE index < highest entry (dir) REP
192                |       index INCR 1 ;
193                |       replace (actual status, pos , status pattern)
194                |   PER .
195                |

```

```

196 actualstatus  |actual status : crowd (index).status .
197                |
198                |ENDPROC status ;
199                |

```

```

200 copy .....|PROC copy (DATASPACE CONST source, TEXT CONST dest name) :
201                |
202                |   IF exists (dest name)
203                |       THEN error (dest name, "existiert bereits")
204                |       ELSE copy file
205                |   FI .
206                |

```

```

207 copyfile     |copy file :
208                |   disable stop ;
209                |   create ( dest name ) ;
210                |   INT VAR index := link (dir, dest name) ;
211                |   IF index > nil
212                |       THEN forget (crowd (index).ds) ;
213                |           crowd (index).ds := source
214                |   FI
215                |
216                |ENDPROC copy ;
217                |

```

```
218 copy .....|PROC copy (TEXT CONST source name, dest name) :  
219 |  
220 |   copy (old (source name), dest name)  
221 |  
222 |ENDPROC copy ;  
223 |
```

```
224 rename .....|PROC rename (TEXT CONST old name, new name) :  
225 |  
226 |   IF exists (new name)  
227 |     THEN error (new name, "existiert bereits")  
228 |   ELIF exists (old name)  
229 |     THEN rename (dir, old name, new name) ;  
230 |         last param (new name)  
231 |   ELSE   error (old name, "gibt es nicht")  
232 |   FI .  
233 |  
234 |ENDPROC rename ;  
235 |  
236 |
```

```
237 beginlist .....|PROC begin list :  
238 |  
239 |   initialize if necessary ;  
240 |   index := 0  
241 |  
242 |ENDPROC begin list ;  
243 |
```

```
244 getlistentry .....|PROC get list entry (TEXT VAR entry, status text) :  
245 |  
246 |   get (dir, entry, index) ;  
247 |   IF found  
248 |     THEN status text := crowd (index).status ;  
249 |     ELSE status text := "" ;  
250 |   FI .  
251 |  
252 |   found  
253 |     found : index > 0 .  
254 |  
255 |ENDPROC get list entry ;  
256 |
```

```
257 writepassword .....|TEXT PROC write password :  
258 |  
259 |   system write password  
260 |  
261 |ENDPROC write password ;  
262 |
```

```
263 readpassword .....|TEXT PROC read password :  
264 |  
265 |   system read password  
266 |  
267 |ENDPROC read password ;  
268 |
```

269

```

270 enterpassword .....|PROC enter password (TEXT CONST password) :
271                       |
272                       |   initialize if necessary ;
273                       |   say ("3"5");
274                       |   INT CONST slash pos := pos (password, "/") ;
275                       |   IF slash pos = 0
276                       |       THEN system write password := password ;
277                       |           system read password := password
278                       |       ELSE system write password := subtext (password, 1, slash pos-1)
279                       |           ;
280                       |           system read password := subtext (password, slash pos+1)
281                       |   FI .
282                       |
283                       |ENDPROC enter password ;

```

```

284 enterpassword .....|PROC enter password (TEXT CONST file name, write pass, read pass) :
285                       |
286                       |   INT CONST index := link (dir, file name) ;
287                       |   IF index > 0
288                       |       THEN set protect password
289                       |       FI .
290
291 setprotectpassword    |set protect password :
292                       |   IF write pass = "" AND read pass = ""
293                       |       THEN crowd (index).protected := FALSE
294                       |       ELSE crowd (index).protected := TRUE ;
295                       |           passwords (index).write := write pass ;
296                       |           passwords (index).read := read pass
297                       |   FI .
298
299                       |ENDPROC enter password ;
300

```

```

301 passwordindex .....|INT PROC password index (TEXT CONST file name) :
302                       |
303                       |   initialize if necessary ;
304                       |   INT CONST index := link (dir, file name) ;
305                       |   IF index > 0 CAND crowd (index).protected
306                       |       THEN index
307                       |       ELSE 0
308                       |   FI
309
310                       |ENDPROC password index ;
311

```

```

312 readpermission .....|BOOL PROC read permission (TEXT CONST name, supply password) :
313                       |
314                       |   (*****
315                       |   (* for reasons of data security the password check algorithm *)
316                       |   (* must not copy parts of the file password into variables *)
317                       |   (* located in the standard dataspace! *)
318                       |   (*****
319
320                       |   access file password ;

```

```

321      | file has no password COR (supply password <> "-" AND read password
+      | match) .
322      |
323  readpasswordmatch | read password match :
324      | file password.read = supply password OR file password.read = "" .
325      |
326  accessfilepassword | access file password :
327      | INT CONST pw index := password index (name) .
328      |
329  filepassword        | file password : passwords (pw index) .
330      |
331  filehasnopassword   | file has no password : pw index = @ .
332      |
333      | ENDPROC read permission ;
334      |
335  writepermission .....| BOOL PROC write permission (TEXT CONST name, supply password) :
336      |
337      | (*****
338      | (* for reasons of data security the password check algorithm *)
339      | (* must not copy parts of the file password into variables *)
340      | (* located in the standard dataspace! *)
341      | (*****
342      |
343      | access file password ;
344      | file has no password COR (supply password <> "-" AND write
+      | password match).
345      |
346  writepasswordmatch  | write password match :
347      | file password.write = supply password OR file password.write = ""
348      |
349  accessfilepassword   | access file password :
350      | INT CONST pw index := password index (name) .
351      |
352  filepassword         | file password : passwords (pw index) .
353      |
354  filehasnopassword    | file has no password : pw index = @ .
355      |
356      | ENDPROC write permission ;
357      |
358  all .....| THESAURUS PROC all :
359      |
360      | initialize if necessary ;
361      | THESAURUS VAR result := dir ; (*ueberflussig ab naechstem
+      | Compiler *)
362      | result
363      |
364      | ENDPROC all ;
365      |

```

Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** local manager

```
366 error .....|PROC error (TEXT CONST file name, error text) :
367             |
368             |   errorstop ("" + file name + "" + error text)
369             |
370             |ENDPROC error ;
371             |
372             |ENDPACKET local manager ;
```

```

1  patternmatch *****|PACKET pattern match DEFINES (* Author:
+                               |P.Heyderhoff *)
2                               |
+                               |
+                               |
3                               |
4                               |
5                               |
6                               |
7                               |
8                               |
9                               |
10                              |
11                              |
12                              |
13                              |
14                              |
15                              |
16                              |
+                               |
+                               |
17                              |
18                              |
19                              |
20                              |
21                              |
22                              |
23                              |
24                              |
25                              |
26                              |
27                              |
28                              |
29                              |
30                              |
31                              |
32                              |
33                              |
34                              |
35                              |
36                              |
37                              |
+                               |
+                               |
38                              |
39                              |
+                               |
+                               |
+                               |
41                              |
+                               |
42                              |

43  - .....|TEXT OP - (TEXT CONST alphabet ):
44          |p:= "";
45          |INT VAR j;
46          |FOR j FROM 0 UPTO 255
47          |REP IF pos(alphabet,code(j)) = 0
48          |THEN p CAT code(j)
49          |FI
50          |PER;
51          |p
52          |ENDOP -;

```

(* Author:
P.Heyderhoff *)
(* Date:
09.06.1986
*)

-,
OR,
**,
any,
notion,
bound,
match,
matchpos,
matchend,
somefix,
UNLIKE,
LIKE :

(*----- Operation codes of the internal intermeadiate language:
-----*)

```

LET
z          = ""0"",
stopz     = ""1""0"",
closez    = ""2""0"",
closor    = ""2""0""3""0"",
or        = ""3"",
oralpha   = ""3""5"",
open2     = ""4""0""4""0"",
alpha     = ""5"",
alphaz    = ""5""0"",
lenz      = ""6""0"",
nilz     = ""6""0""0""0""7""0"", (* = any (0) *)
starz     = ""7""0"",
star      = ""8""0""2""7""0""1""0"", (* = any ** 1 *)
powerz    = ""8""0"",
powerz0   = ""8""0""1"",
notionz   = ""9""0"",
fullz     = ""10""0"",
boundz    = ""11""0"";

```

```

(*-----*)
LET undefined = 0, (* fixleft
value        = *, (* value
forcer       = 0, (* value
parameter    *)
delimiter = " !""#$%&'()*+,-./:;<=>?@^`~-"; (* for 'PROC
notion' *)

```

```

53      |
54  OR .....|TEXT OP OR (TEXT CONST a, b):
55      |    open2 + notnil (a) + closor + notnil (b) + closez
56      |    ENDOP OR;
57      |
58  ** .....|TEXT OP ** (TEXT CONST p, INT CONST x):
59      |    powerz + code (1+x) + notnil (p) + stopz
60      |    ENDOP **;
61      |
62      |TEXT CONST any:= starz;
63      |
64  any .....|TEXT PROC any (INT CONST n):
65      |    TEXT VAR t:= " ";
66      |    replace (t, 1, ABSn);
67      |    lenz + t + starz
68      |    ENDPROC any;
69      |
70  any .....|TEXT PROC any (TEXT CONST a): alphaz + a + starz ENDPROC any;
71      |
72  any .....|TEXT PROC any (INT CONST n, TEXT CONST a):
73      |    TEXT VAR t:= " ";
74      |    replace (t, 1, ABSn);
75      |    lenz + t + alphaz + a + starz
76      |    ENDPROC any;
77      |
78  notion .....|TEXT PROC notion (TEXT CONST t): notionz + notnil(t) + stopz ENDPROC
79  +          |    notion;
80  notnil .....|TEXT PROC notnil (TEXT CONST t):
81      |    IF t = ""
82      |    THEN nilz
83      |    ELSE t
84      |    FI
85      |    ENDPROC notnil;
86      |
87      |TEXT CONST bound := boundz;
88      |
89  full .....|TEXT PROC full (TEXT CONST t): fullz + t + stopz ENDPROC full;
90      |
91  match .....|TEXT PROC match (INT CONST x):
92      |    subtext (p, matchpos(x), matchend(x))
93      |    ENDPROC match;
94      |

```

```

95 matchpos .....|INT PROC matchpos (INT CONST x): mapos (1 + x MOD 256) ENDPROC
+                |matchpos;
96                |
97 matchend .....|INT PROC matchend (INT CONST x): maend (1 + x MOD 256) - 1
98                |ENDPROC matchend;
99                |
100               |(*----- GLOBAL VARIABLES:
+                |-----*)
101               |
102               |ROW 256 INT VAR
103               |      (* Table of match registers. Each entry consists of
+                |      two *)
104               |      (* pointers, which points to the TEXT object 't'
+                |      *)
105               |      mapos,      (* points to the beginning of the match
+                |      *)
106               |      maend;      (* points to the position after the end of match
+                |      *)
107               |
108               |INT VAR ppos, tpos, (* workpositions in pattern 'p' and text 't'
+                |      *)
109               |      floatpos, (* accumulation of all pending floatlengths
+                |      *)
110               |      failpos, (* result of 'PROC in alpha'
+                |      *)
111               |      plen, tlen, (* length of pattern 'p' and length of text
+                |      't' *)
112               |      skipcount, (* for track forward skipping
+                |      *)
113               |      multi, vari; (* for handling of nonexclusive alternatives
+                |      *)
114               |
115               |TEXT VAR p,      (* the pattern to be find or some result
+                |      *)
116               |      stack,      (* stack of pending assignments
+                |      *)
117               |      alphabet:=""; (* result of 'PROC find alpha', reset to nil
+                |      *)
118               |      (* after its usage by 'find any'
+                |      *)
119               |
120               |BOOL VAR fix,      (* text position is fixed and not floating
+                |      *)
121               |      no vari; (* not varying the order of alternatives
+                |      *)
122               |
123 somefix .....|TEXT PROC somefix (TEXT CONST pattern):
124               |
125               |      (* delivers the first text occuring unconditionally in the
+                |      pattern *)
126               |
127               |      p:= pattern;
128               |      INT VAR j:= 1, n:= 0, k, len:= LENGTH p;
129               |      REP
130               |      SELECT text( subtext (p, j, j+1), 2) ISUB 1 OF
131               |      CASE 1,3,7,9,10,11: j INCR 2
132               |      CASE 2:          j INCR 2; n DECR 1 (* condition closed
+                |      *)

```

```

133          CASE 4:          j INCR 2; n INCR 1 (* condition opened
+          *)
134          CASE 5:          j := pos (p, starz, j+2) + 2
135          CASE 6:          j INCR 4
136          CASE 8:          j INCR 3
137          OTHERWISE      k:= pos(p, z, j+1) - 1;
138                          IF k <= 0 THEN k:= 1+len FI;
139                          IF star found
140                          THEN change (p, starpos, starpos, star);
141                              len:= LENGTH p;
142                              k:= starpos
143                          FI;
144                          IF n = 0 CAND ( p SUB k ) <> or CAND k > j
145                          THEN LEAVE somefix WITH subtext(p,j,k-1)
146                          ELSE j:=k
147                          FI
148          ENDSELECT
149          UNTIL j > len
150          PER;
151          ""
152
153          starfound
154          star found:
155          INT VAR starpos:= pos (p, "", j);
156          starpos > 0 CAND starpos <= k .
157          ENDPROC somefix;
158

```

```

159          skip ..... PROC skip (TEXT CONST p, BOOL CONST upto or):
160
161          (* skips 'ppos' upto the end of the opened nest, n = nesting
+          level *)
162
163          INT VAR n:= 0;
164          REP
165          SELECT text (subtext (p, ppos, ppos+1), 2) ISUB 1 OF
166          CASE 1,2:      IF n = 0
167                          THEN LEAVE skip
168                          FI;
169                          ppos INCR 2;
170                          nDECR1
171          CASE 3:      IF n = 0 CAND upto or
172                          THEN LEAVE skip
173                          FI;
174                          ppos INCR 2
175          CASE 7:      ppos INCR 2
176          CASE 4,9,10,11: ppos INCR 2;
177                          n INCR 1
178          CASE 5:      ppos:= pos (p, starz, ppos+2) + 2
179          CASE 6:      ppos INCR 4
180          CASE 8:      ppos INCR 3;
181                          n INCR 1
182          OTHERWISE  ppos:= pos(p, z, ppos+1) - 1;
183                          IF ppos < 0
184                          THEN ppos:= plen;
185                          LEAVE skip
186                          FI
187          ENDSELECT
188          PER
189          ENDPROC skip;

```

190

191 UNLIKE| BOOL OP UNLIKE (TEXT CONST t, p): NOT (t LIKE p) ENDOP UNLIKE;
 192

193 LIKE| BOOL OP LIKE (TEXT CONST t, pattern):

194 | init;
 195 | BOOL CONST found:= find (t,1,1, fixresult, floatresult);
 196 | save;
 197 | found.
 198

199 init

```

init:  no vari:= TRUE;
       vari:= 0;
       tlen:= 1 + LENGTH t;
       p:= full (pattern);
       IF pos (p, bound) > 0
       THEN
           IF subtext (p, 14, 15) = bound
           THEN p:= subtext (p, 1, 8) + powerz0 + subtext (p,
               +
               16)
           FI;
           plen:= LENGTH p - 7;
           IF subtext (p, plen, plen+1) = bound
           THEN p:= subtext (p, 1, plen - 1) + stopz + stopz
           FI;
       FI;
       plen:= LENGTH p + 1;
       INT VAR fixresult, floatresult;
       tpos:= 1;
       floatpos:= 0;
       stack:= "";
       alphabet:= "";
       fix:= TRUE;
       skipcount:= 0;
       multi:= 0.
    
```

223 save

```

save:  p:= t
       ENDOP LIKE;
    
```

227 | (*----- Realisation of the pattern matching algorithms 'find'
 + | -----*)
 228

229 find| BOOL PROC find

230 | (TEXT CONST t, INT CONST unit, from, INT VAR fixleft,
 + | floatlen):

```

231                   |
232                   |     initialize;
233                   |     BOOL CONST found:= pattern unit;
234                   |     SELECT next command * unit OF
235                   |         CASE 0,1,2:         found
236                   |         CASE 3:             next;
237                   |                             find alternative
238                   |         OTHERWISE         find concatenation
239                   |     ENDSELECT .
240
    
```

```

241 findalternative      find alternative:
242                     IF found
243                       THEN save left position;
244                       backtrack;
245                       IF find pattern CAND better
246                         THEN note multiplicity
247                         ELSE back to first one
248                         FI
249                       ELSE backtrack multi
250                       FI.
251
252 better               better:                permutation XOR more left.
253
254 permutation         permutation:            vari MOD 2 = 1.
255
256 saveleftposition    save left position:      j:= fixleft.
257
258 moreleft            more left:              j > fixleft.
259
260 backtrackmulti      backtrack multi:        multi:= 2 * backmulti + 1;
261                     vari:= backvari DIV 2;
262                     find pattern.
263
264 notemultiplicity    note multiplicity:      multi:= 2 * multi + 1;
265                     vari:= vari DIV 2;
266                     TRUE.
267
268 backtofirstone      back to first one:      backtrack;
269                     IF find first subpattern
270                       THEN skip (p, FALSE);
271                       note multiplicity
272                       ELSE errorstop ("pattern");
273                       FALSE
274                       FI.
275
276 findconcatenation   find concatenation:
277                     IF found
278                       THEN IF ppos=plen COR find pattern COR track forward
279                           COR ( multi > backmulti CAND vari = 0 CAND find variation
280                               )
281                           THEN TRUE
282                           ELSE backtrack; FALSE
283                           FI
284                       ELSE skip (p, TRUE); FALSE
285                       FI.
286
286 trackforward        track forward:          (* must be performed before
+                    variation *)
287                     j:=0;
288                     last multi:= multi;
289                     last vari:= vari;
290                     WHILE skipcount = 0
291                     REP IF tlen = tpos

```

```

292                                     THEN LEAVE track forward WITH FALSE
293                                     FI;
294                                     backtrack;
295                                     j INCR 1;
296                                     skipcount:= j
297                                     UNTIL find first subpattern CAND find pattern
298                                     PER;
299                                     j:= skipcount;
300                                     skipcount:=0;
301                                     j=0.
302
303     findvariation                       find variation:
304                                         multi:= last multi;
305                                         vari:= last vari;
306                                         FOR k FROM 1 UPTO (multi+1) DIV (backmulti+1) - 1
307                                         REP backtrack with variation;
308                                         IF find first subpattern CAND find pattern
309                                         THEN vari:=0;
310                                         LEAVE find variation WITH TRUE
311                                     FI
312                                     PER;
313                                     FALSE.
314
315     backtrackwithvariation              backtrack with variation:
316                                         backtrack;
317                                         vari:= k.
318
319     findpattern                          find pattern:
320                                         find (t, 1, ppos+forcer, fixresult, floatresult) CAND keep
321     +                                     result.
322
322     findfirstsubpattern                  find first subpattern:
323                                         find (t, 0, from, fixresult, floatresult) CAND keep result
324
325     initialize                            initialize:
326                                         INT VAR    j,
327                                         k,
328                                         fixresult,
329                                         floatresult,
330                                         last multi,
331                                         last vari;
332                                         BOOL CONST backfix:= fix;
333                                         TEXT CONST backstack:= stack;
334                                         floatlen:= 0;
335                                         INT CONST back:= tpos,
336                                         backfloat:= floatpos,
337                                         backskip:= skipcount,
338                                         backmulti:= multi,
339                                         backvari:= vari;
340                                         fixleft:= fixleft0.
341
342     fixleft0                              fixleft0: IF fix THEN back ELSE undefined FI.
343

```

```

344 backtrack | backtrack:
345 |     fix:= backfix;
346 |     tpos:= back;
347 |     fixleft:= fixleft0;
348 |     floatlen:= 0;
349 |     floatpos:= backfloat;
350 |     stack:= backstack;
351 |     skipcount:= backskip;
352 |     multi:= backmulti;
353 |     vari:= backvari.
354 |
355 | keepresult | keep result:
356 | | IF fixleft = undefined
357 | | THEN IF fixresult = undefined
358 | |     THEN floatlen INCR floatresult
359 | |     ELSE fixleft := fixresult - floatlen;
360 | |         floatpos DECR floatlen;
361 | |         floatlen:= 0
362 | |     FI
363 | | FI;
364 | | TRUE.
365 |
366 | patternunit | pattern unit:
367 | | init ppos;
368 | | SELECT command OF
369 | |     CASE 1,2: find end
370 | |     CASE 3:   find nil
371 | |     CASE 4:   find choice
372 | |     CASE 5:   find alphabet
373 | |     CASE 6:   find fixlength any
374 | |     CASE 7:   find varlength any
375 | |     CASE 8:   find and store match
376 | |     CASE 9:   find notion
377 | |     CASE 10:  find full
378 | |     CASE 11:  next; find nil
379 | |     OTHERWISE find plain text
380 | |         SELECT.
381 | |
382 | |
383 | |
384 | |
385 | |
386 | |
387 | |
388 | |
389 | |
390 | |
391 | |
392 | |
393 | |
394 | |
395 | |
396 | |
397 | |
398 | |
399 | |
400 | |
401 | |
402 | |
403 | |
404 | |
405 | |
406 | |
407 | |
408 | |
409 | |
410 | |
411 | |
412 | |
413 | |
414 | |
415 | |
416 | |
417 | |
418 | |
419 | |
420 | |
421 | |
422 | |
423 | |
424 | |
425 | |
426 | |
427 | |
428 | |
429 | |
430 | |
431 | |
432 | |
433 | |
434 | |
435 | |
436 | |
437 | |
438 | |
439 | |
440 | |
441 | |
442 | |
443 | |
444 | |
445 | |
446 | |
447 | |
448 | |
449 | |
450 | |
451 | |
452 | |
453 | |
454 | |
455 | |
456 | |
457 | |
458 | |
459 | |
460 | |
461 | |
462 | |
463 | |
464 | |
465 | |
466 | |
467 | |
468 | |
469 | |
470 | |
471 | |
472 | |
473 | |
474 | |
475 | |
476 | |
477 | |
478 | |
479 | |
480 | |
481 | |
482 | |
483 | |
484 | |
485 | |
486 | |
487 | |
488 | |
489 | |
490 | |
491 | |
492 | |
493 | |
494 | |
495 | |
496 | |
497 | |
498 | |
499 | |
500 | |
501 | |
502 | |
503 | |
504 | |
505 | |
506 | |
507 | |
508 | |
509 | |
510 | |
511 | |
512 | |
513 | |
514 | |
515 | |
516 | |
517 | |
518 | |
519 | |
520 | |
521 | |
522 | |
523 | |
524 | |
525 | |
526 | |
527 | |
528 | |
529 | |
530 | |
531 | |
532 | |
533 | |
534 | |
535 | |
536 | |
537 | |
538 | |
539 | |
540 | |
541 | |
542 | |
543 | |
544 | |
545 | |
546 | |
547 | |
548 | |
549 | |
550 | |
551 | |
552 | |
553 | |
554 | |
555 | |
556 | |
557 | |
558 | |
559 | |
560 | |
561 | |
562 | |
563 | |
564 | |
565 | |
566 | |
567 | |
568 | |
569 | |
570 | |
571 | |
572 | |
573 | |
574 | |
575 | |
576 | |
577 | |
578 | |
579 | |
580 | |
581 | |
582 | |
583 | |
584 | |
585 | |
586 | |
587 | |
588 | |
589 | |
590 | |
591 | |
592 | |
593 | |
594 | |
595 | |
596 | |
597 | |
598 | |
599 | |
600 | |
601 | |
602 | |
603 | |
604 | |
605 | |
606 | |
607 | |
608 | |
609 | |
610 | |
611 | |
612 | |
613 | |
614 | |
615 | |
616 | |
617 | |
618 | |
619 | |
620 | |
621 | |
622 | |
623 | |
624 | |
625 | |
626 | |
627 | |
628 | |
629 | |
630 | |
631 | |
632 | |
633 | |
634 | |
635 | |
636 | |
637 | |
638 | |
639 | |
640 | |
641 | |
642 | |
643 | |
644 | |
645 | |
646 | |
647 | |
648 | |
649 | |
650 | |
651 | |
652 | |
653 | |
654 | |
655 | |
656 | |
657 | |
658 | |
659 | |
660 | |
661 | |
662 | |
663 | |
664 | |
665 | |
666 | |
667 | |
668 | |
669 | |
670 | |
671 | |
672 | |
673 | |
674 | |
675 | |
676 | |
677 | |
678 | |
679 | |
680 | |
681 | |
682 | |
683 | |
684 | |
685 | |
686 | |
687 | |
688 | |
689 | |
690 | |
691 | |
692 | |
693 | |
694 | |
695 | |
696 | |
697 | |
698 | |
699 | |
700 | |
701 | |
702 | |
703 | |
704 | |
705 | |
706 | |
707 | |
708 | |
709 | |
710 | |
711 | |
712 | |
713 | |
714 | |
715 | |
716 | |
717 | |
718 | |
719 | |
720 | |
721 | |
722 | |
723 | |
724 | |
725 | |
726 | |
727 | |
728 | |
729 | |
730 | |
731 | |
732 | |
733 | |
734 | |
735 | |
736 | |
737 | |
738 | |
739 | |
740 | |
741 | |
742 | |
743 | |
744 | |
745 | |
746 | |
747 | |
748 | |
749 | |
750 | |
751 | |
752 | |
753 | |
754 | |
755 | |
756 | |
757 | |
758 | |
759 | |
760 | |
761 | |
762 | |
763 | |
764 | |
765 | |
766 | |
767 | |
768 | |
769 | |
770 | |
771 | |
772 | |
773 | |
774 | |
775 | |
776 | |
777 | |
778 | |
779 | |
780 | |
781 | |
782 | |
783 | |
784 | |
785 | |
786 | |
787 | |
788 | |
789 | |
790 | |
791 | |
792 | |
793 | |
794 | |
795 | |
796 | |
797 | |
798 | |
799 | |
800 | |
801 | |
802 | |
803 | |
804 | |
805 | |
806 | |
807 | |
808 | |
809 | |
810 | |
811 | |
812 | |
813 | |
814 | |
815 | |
816 | |
817 | |
818 | |
819 | |
820 | |
821 | |
822 | |
823 | |
824 | |
825 | |
826 | |
827 | |
828 | |
829 | |
830 | |
831 | |
832 | |
833 | |
834 | |
835 | |
836 | |
837 | |
838 | |
839 | |
840 | |
841 | |
842 | |
843 | |
844 | |
845 | |
846 | |
847 | |
848 | |
849 | |
850 | |
851 | |
852 | |
853 | |
854 | |
855 | |
856 | |
857 | |
858 | |
859 | |
860 | |
861 | |
862 | |
863 | |
864 | |
865 | |
866 | |
867 | |
868 | |
869 | |
870 | |
871 | |
872 | |
873 | |
874 | |
875 | |
876 | |
877 | |
878 | |
879 | |
880 | |
881 | |
882 | |
883 | |
884 | |
885 | |
886 | |
887 | |
888 | |
889 | |
890 | |
891 | |
892 | |
893 | |
894 | |
895 | |
896 | |
897 | |
898 | |
899 | |
900 | |
901 | |
902 | |
903 | |
904 | |
905 | |
906 | |
907 | |
908 | |
909 | |
910 | |
911 | |
912 | |
913 | |
914 | |
915 | |
916 | |
917 | |
918 | |
919 | |
920 | |
921 | |
922 | |
923 | |
924 | |
925 | |
926 | |
927 | |
928 | |
929 | |
930 | |
931 | |
932 | |
933 | |
934 | |
935 | |
936 | |
937 | |
938 | |
939 | |
940 | |
941 | |
942 | |
943 | |
944 | |
945 | |
946 | |
947 | |
948 | |
949 | |
950 | |
951 | |
952 | |
953 | |
954 | |
955 | |
956 | |
957 | |
958 | |
959 | |
960 | |
961 | |
962 | |
963 | |
964 | |
965 | |
966 | |
967 | |
968 | |
969 | |
970 | |
971 | |
972 | |
973 | |
974 | |
975 | |
976 | |
977 | |
978 | |
979 | |
980 | |
981 | |
982 | |
983 | |
984 | |
985 | |
986 | |
987 | |
988 | |
989 | |
990 | |
991 | |
992 | |
993 | |
994 | |
995 | |
996 | |
997 | |
998 | |
999 | |
1000 | |

```

397

398 findchoice

```
find choice: IF find pattern
              THEN next; TRUE
              ELSE next; FALSE
              FI.
```

399
400
401
402

403 findplaintext

```
find plain text: find text upto next command;
                  IF fix THEN allow fix position only
                  ELIF text found THEN allow variable position
                  ELSE allow backtrack
                  FI.
```

404
405
406
407
408

409 findtextuptonextcommand

```
find text upto next command:
  ppos:= pos (p, z, from + 1);
  IF ppos = 0
  THEN ppos:= plen
  ELSE ppos DECR 1
  FI;
  IF star found
  THEN change (p, starpos, starpos, star);
  plen:= 1 + LENGTH p;
  ppos:= starpos
  FI;
  tpos:= pos (t, subtext (p, from, ppos - 1), tpos).
```

410
411
412
413
414
415
416
417
418
419
420
421

422 starfound

```
star found:
  INT VAR starpos:= pos (p, "*", from);
  starpos > 0 CAND starpos <= ppos .
```

423
424
425

426 textfound

```
text found:
  WHILE skipcount > 0 CAND tpos > 0
  REP skipcount DECR 1;
  tpos:= pos (t, subtext(p,from,ppos-1), tpos+1)
  PER;
  tpos > 0 .
```

427
428
429
430
431
432

433 allowfixpositiononly

```
allow fix position only:
  IF tpos = back
  THEN tpos INCR (ppos-from); TRUE
  ELSE tpos:= back;
  from = ppos
  FI.
```

434
435
436
437
438
439

440 allowvariableposition

```
allow variable position:
  IF alphabet = "" COR in alpha (t, back, tpos)
  THEN fix it;
  tpos INCR (ppos-from);
  TRUE
  ELSE tpos:= back;
  FALSE
  FI.
```

441
442
443
444
445
446
447
448

<p>449 allowbacktrack 450 451 452 453 454 455 456</p> <p>457 findalphabet 458 459 460 461 462</p> <p>463 findfixlengthany 464 465 466 467 468 469 470</p> <p>471 getlengthvalue 472 473 474</p> <p>475 findalphaattribut 476 477 478 479</p> <p>480 findanywithfixlength 481 482 483 484 485 486 487 488 + 489 490 491 492 493</p> <p>494 findanyinalphabetwithf 495 496 497 498 499 500</p>	<p>allow backtrack: tpos:= back; IF from = ppos THEN fix it; TRUE ELSE FALSE FI .</p> <p>find alphabet: j:= pos (p, starz, ppos); alphabet:= subtext (p, ppos, j-1); ppos := j; TRUE.</p> <p>find fixlength any: get length value; find alpha attribut; IF alphabet = "" THEN find any with fix length ELSE find any in alphabet with fix length FI.</p> <p>get length value: floatlen:= subtext(p, ppos, ppos+1) ISUB 1; ppos INCR 4.</p> <p>find alpha attribut: IF (p SUB (ppos-2)) = alpha CAND find alphabet THEN next FI.</p> <p>find any with fix length: tpos INCR floatlen; IF tpos > tlen THEN tpos:= back; floatlen:=0; FALSE ELSE IF fix THEN floatlen:= 0 ELIF floatlen = 0 THEN fix it (* unlike niltext 6.6. *) ELSE floatpos INCR floatlen FI; TRUE FI.</p> <p>find any in alphabet with fix length: IF first character in alpha THEN IF NOT fix THEN fix it FI; set fix found ELSE set fix not found FI.</p>
--	---

```

501 firstcharacterinalpha first character in alpha:
502 (fix COR advance) CAND in alpha (t, tpos, tpos+floatlen).
503
504 advance advance:
505 FOR tpos FROM back UPTO tlen
506 REP IF pos (alphabet, t SUB tpos) > 0
507 THEN LEAVE advance WITH TRUE
508 FI
509 PER;
510 FALSE.
511
512 fixit fix it:
513 fixleft:= back-floatpos;
514 make fix (back);
515 fixleft:= tpos.
516
517 setfixfound set fix found:
518 tpos INCR floatlen;
519 floatlen:= 0;
520 alphabet:= "";
521 TRUE.
522
523 setfixnotfound set fix not found:
524 tpos:= back;
525 alphabet:= "";
526 floatlen:= 0;
527 FALSE.
528
529 findvarlengthany find varlength any:
530 IF alphabet = ""
531 THEN really any
532 ELSE find varlength any in alphabet
533 FI.
534
535 reallyany really any:
536 IF fix
537 THEN fix:= FALSE;
538 fixleft:= tpos
539 ELIF floatpos = 0
540 THEN fixleft:= tpos (* 6.6.
541 *)
542 +
543 FI;
544 TRUE .
545
546 findvarlengthanyinalph find varlength any in alphabet:
547 IF fix THEN fixleft := tpos FI;
548 IF fix CAND pos (alphabet, t SUB tpos) > 0
549 COR NOT fix CAND advance
550 THEN IF NOT fix THEN fix it FI;
551 set var found
552 ELSE set var not found
553 FI.
554
555 setvarfound set var found:
556 tpos:= end of varlength any;
557 alphabet:= "";
558 TRUE.

```

```

553   setvarnotfound      | set var not found:   tpos:= back;
554                               alphabet:= "";
555                               FALSE.

556   endofvarlengthany   | end of varlength any: IF NOT in alpha(t,tpos,tlen)
557                               THEN failpos
558                               ELSE tlen
559                               FI.
560

561   findandstorematch    | find and store match: get register name;
562                               IF find pattern
563                               THEN next;
564                               store;
565                               TRUE
566                               ELSE next;
567                               FALSE
568                               FI.
569

570   store                | store:               IF fix
571                               THEN mapos (reg):= fixleft;
572                               maend (reg):= tpos
573                               ELSE stack CAT code(floatlen) +
574                               code(floatpos) +
575                               +
576                               code(fixleft) + c
577                               FI.

577   getregistername      | get register name:   TEXT CONST c:= p SUB (ppos);
578                               INT VAR reg:= code (c);
579                               ppos INCR 1.
580

581   findnotion           | find notion:         float notion;
582                               exhaust notion .
583

584   floatnotion          | float notion:        j:= back;
585                               REP IF find pattern
586                               THEN IF is notion (t, fixleft)
587                               THEN LEAVE find notion WITH TRUE
588                               ELIF backfix
589                               THEN LEAVE float notion
590                               ELSE go ahead FI
591                               ELIF j=back
592                               THEN next;
593                               LEAVE find notion WITH FALSE
594                               ELSE LEAVE float notion
595                               FI
596                               PER.
597

598   goahead              | go ahead:            j INCR 1;
599                               IF simple THEN j:= max (tpos, j) FI;
600                               notion backtrack.
601

602   simple               | simple:              k:= from;
603                               REP k := pos (p, z, k+2);
604                               IF k > ppos-3
605                               THEN LEAVE simple WITH TRUE

```

```

606                                     ELIF pos (oralpha, p SUB k-1) > 0
607                                     THEN LEAVE simple WITH FALSE
608                                     FI
609                                     PER;
610                                     FALSE.
611
612  notionbacktrack                    notion backtrack: tpos:= j;
613                                     fix:= backfix;
614                                     fixleft:= fixleft0;
615                                     floatlen:= 0;
616                                     floatpos:= backfloat + tpos - back;
617                                     stack:= backstack;
618                                     ppos:= from + 2 .
619
620  exhaustnotion                      exhaust notion: IF  notion expansion
621                                     COR multi > backmulti
622                                     CAND no vari
623                                     CAND notion variation
624                                     THEN TRUE
625                                     ELSE backtrack; FALSE
626                                     FI.
627
628  notionexpansion                    notion expansion: j:= 0;
629                                     multi:= last multi;
630                                     vari:= last vari;
631                                     WHILE skipcount = 0
632                                     REP skip and try PER;
633                                     j:= skipcount;
634                                     skipcount:= 0;
635                                     j = 0.
636
637  skipandtry                         skip and try:  backtrack;
638                                     j INCR 1;
639                                     skipcount:=j;
640                                     ppos:= from + 2;
641                                     IF  find pattern
642                                     THEN IF  is notion (t, fixleft)
643                                             THEN LEAVE find notion WITH TRUE
644                                             FI
645                                     ELSE next; LEAVE find notion WITH FALSE
646                                     FI .
647
648  notionvariation                    notion variation: no vari:= FALSE;
649                                     last multi:= multi;
650                                     last vari:= vari;
651                                     FOR k FROM 1 UPTO (multi+1) DIV (backmulti+1) - 1
652                                     REP backtrack with variation;
653                                     IF  find first subpattern
654                                     THEN no vari:= TRUE;
655                                             LEAVE find notion WITH TRUE
656                                     FI
657                                     PER;
658                                     no vari:= TRUE;
659                                     FALSE.
660

```

```

661 findfull find full:
662 find pattern CAND (end of line COR exhaust line).
663
664 endofline end of line:
665 next;
666 IF fix
667 THEN tpos = tlen
668 ELSE tpos:= tlen;
669 make fix (1);
670 TRUE
671 FI.
672
673 exhaustline exhaust line:
674 IF full expansion COR multi > 0 CAND no vari CAND full
+ variation
675 THEN TRUE ELSE backtrack;
676 FALSE
677 FI.
678
679 fullexpansion full expansion:
680 j:=0;
681 last multi:= multi;
682 last vari:= vari;
683 WHILE skipcount = 0
684 REP IF tlen = tpos
685 THEN LEAVE full expansion WITH FALSE
686 FI;
687 backtrack;
688 j INCR 1;
689 skipcount:= j;
690 ppos:=from + 2
691 UNTIL find pattern CAND tpos=tlen
692 PER;
693 j:= skipcount;
694 skipcount:=0;
695 j=0.
696
697 fullvariation full variation:
698 no vari:= FALSE;
699 multi:= last multi;
700 vari:= last vari;
701 FOR k FROM 1 UPTO multi
702 REP backtrack with variation;
703 IF find first subpattern
704 THEN no vari:= TRUE;
705 LEAVE find WITH TRUE
706 FI
707 PER;
708 no vari:= TRUE;
709 FALSE.
710
711 ENDPROC find;
712
713 isnotion .....|BOOL PROC is notion (TEXT CONST t, INT CONST fixleft):
714 ppos INCR 2;
715 ( NOT fix

```

```

716                                     COR tpos = tlen
717                                     COR pos (delimiter, t SUB tpos) > 0
718                                     COR pos (delimiter, t SUB tpos-1) > 0
719                                     COR (t SUB tpos) <= "Z"
720                                     CAND (t SUB tpos-1) > "Z" )
721                                     CAND (   fixleft <= 1
722                                     COR pos (delimiter, t SUB fixleft-1) > 0
723                                     COR pos (delimiter, t SUB fixleft) > 0
724                                     COR (t SUB fixleft) > "Z"
725                                     CAND (t SUB fixleft-1) <= "Z" )
727                                     END PROC is notion;
728
729 makefix .....|PROC make fix (INT CONST back):
730                                     WHILE stack not empty
731                                     REP INT VAR reg:= code (stack SUB top),
732                                     pos:= code (stack SUB top-1),
733                                     len:= code (stack SUB top-3),
734                                     dis:= code (stack SUB top-2) - floatpos;
735                                     maend(reg):= min (tpos + dis, tlen);      (* 6.6.
736                                     *)
737                                     mapos(reg):= pos or fix or float;
738                                     stack:= subtext (stack,1,top-4)
739                                     PER;
740                                     fix:= TRUE;
741                                     floatpos:= 0 .
742
743 stacknotempty |stack not empty: INT VAR top:= LENGTH stack;
744                                     top > 0.
745
746 posorfixorfloat |pos or fix or float:
747                                     IF pos = undefined
748                                     THEN IF len = 0
749                                     THEN min (back + dis, tlen)
750                                     ELSE maend(reg) - len
751                                     FI
752                                     ELSE pos
753                                     FI.
754
755                                     ENDPROC make fix;
756
757 inalpha .....|BOOL PROC in alpha (TEXT CONST t, INT CONST from, to):
758                                     FOR failpos FROM from UPTO to - 1
759                                     REP IF pos (alphabet, t SUB failpos) = 0
760                                     THEN LEAVE in alpha WITH FALSE
761                                     FI
762                                     PER;
763                                     TRUE
764                                     ENDPROC in alpha;
765
766 notion .....|TEXT PROC notion (TEXT CONST t, INT CONST r): notion (t) ** r
767                                     ENDPROC notion;
768                                     ENDPACKET pattern match;

```

```
1 | (* ----- VERSION 35 02.06.86 ----- *)
2 | filehandling ***** PACKET file handling DEFINES (* Autoren: J.Liedtke, D.Martinek *)
3 | | (*****)
```

4		
5		FILE,
6		:=,
7		sequential file,
8		reorganize,
9		input,
10		output,
11		modify,
12		close,
13		putline,
14		getline,
15		put,
16		get,
17		write ,
18		line,
19		reset,
20		down,
21		up,
22		downety,
23		uppety,
24		pattern found,
25		to first record,
26		to line,
27		to eof,
28		insert record,
29		delete record,
30		read record,
31		write record,
32		is first record,
33		eof,
34		line no,
35		FRANGE,
36		set range,
37		reset range ,
38		remove,
39		clear removed,
40		reinsert,
41		max line length,
42		edit info,
43		line type ,
44		copy attributes ,
45		headline,
46		put tabs,
47		get tabs,
48		col,
49		word,
50		at,
51		removed lines,
52		exec,
53		pos ,
54		len ,
55		subtext ,
56		change ,
57		lines ,
58		segments ,
59		mark ,
60		mark line no ,
61		mark col ,
62		set marked range ,

```

63      |      split line ,
64      |      concatenate line ,
65      |      prefix ,
66      |      sort ,
67      |      lexsort :
68      |
69      |
70      | (*****
71      |      **)
72      | (*
73      |      *)
74      | (* Terminologie:
75      |      *)
76      | (*
77      |      *)
78      | (*
79      |      *)
80      | (*
81      |      *)
82      | (*
83      |      *)
84      | (*
85      |      *)
86      | (*
87      |      *)
88      | (*
89      |      *)
90      | (*
91      |      *)
92      | (*
93      |      *)
94      | (*
95      |      *)
96      | (*
97      |      *)
98      | (*
99      |      *)

```

```

      split line ,
      concatenate line ,
      prefix ,
      sort ,
      lexsort :

(*****
**)
(*
*)
(* Terminologie:
*)
(*
*)
(*
*)
(* ATOMROW      Menge aller Atome eines FILES.
*)
(*
*)
(*      Die einzelnen Atome haben zwar eine Position
*)
(*      im Row, aber in dieser Betrachtung keine
*)
(*      logische Reihenfolge.
*)
(*
*)
(*
*)
(* ATOM      Basiselement, kann eine Zeile der Datei und die
*)
(*      zugehoerige Verwaltungsinformation aufnehmen
*)
(*
*)
(* CHAIN      Zyklisch geschlossene Kette von Segmenten.
*)
(*
*)
(* SEGMENT      Teilbereich des Atomrows, enthaelt 1 oder mehr
*)
(*      zusammenhaengende Atoms.
*)
(*
*)
(*      Jedes Segment hat ein Vorgaenger- und ein
*)
(*      Nachfolgersegment.
*)
(*
*)
(*      Jedes Segment enthaelt einen logisch zusammen-
*)
(*      haengenden Teile einer Sequence.
*)
(*
*)
(*
*)
(* SEQUENCE      Logische Folge von Lines.
*)
(*
*)
(*      Jede Sequence ist Teil einer Chain oder besteht
*)
(*      vollstaendig daraus:
*)
(*
*)
(*
*)
(*      SEG1--SEG2--SEG3--SEG4--SEG5
*)
(*
*)
(*      :----sequence----:
*)

```

```

98      |(* *)
99      |(* *)           Die 'Reihenfolge' ebenso wie die 'Anzahl' der
100     |(* *)           Lines ist eine wesentliche Eigenschaft einer
101     |(* *)           Sequence.
102     |(* *)
103     |(* LINE       Ein Atom als Element ein Sequence betrachtet.
104     |(* *)
105     |(* *)
106     |*****
107     |(**)
108     |(* Eigenschaften:
109     |(* *)
110     |(* Folgende Mengen bilden eine Zerlegung (im math. Sinn) einer
111     |(* *)
112     |(* *)           gesamtan Datei:
113     |(* *)           used segment chain
114     |(* *)           scratch segment chain
115     |(* *)           free segment chain
116     |(* *)           unused tail
117     |(* *)
118     |(* *)           Fuer jedes X aus (used, scratch, free) gelten:
119     |(* *)           'X sequence' ist echte Teilmenge von 'X segment chain'.
120     |(* *)
121     |(* *)           (Daraus folgt, es gibt keine leere
122     |(* *)           'chain'.) *)
123     |(* *)           'X segment chain' ist zyklisch gekettet.
124     |(* *)
125     |(* *)           Alle Atome von 'X segment chain' haben definierten Inhalt.
126     |(* *)
127     |*****
128     |(**)
129     |

```

```

130      |LET file size  = 4075 ,
131      |           nil      = 0 ,
132      |
133      |           free root = 1 ,
134      |           scratch root = 2 ,
135      |           used root  = 3 ,
136      |           first unused = 4 ;
137
138
139      |LET SEQUENCE = STRUCT (INT index, segment begin, segment end,
140      |                       INT line no, lines),
141      |           SEGMENT = STRUCT (INT succ, pred, end),
142      |           ATOM    = STRUCT (SEGMENT seg, INT type, TEXT line),
143      |           ATOMROW = ROW filesize ATOM,
144
145      |           LIST    = STRUCT (SEQUENCE used, INT prefix lines, postfix
146      |                               lines,
147      |                               SEQUENCE scratch, free, INT unused tail,
148      |                               INT mode, col, limit, edit info, mark line,
149      |                               mark col,
150      |                               ATOMROW atoms);
151
152      |TYPE FILE = BOUND LIST ;
153
154      |TYPE FRANGE = STRUCT (INT pre, post, BOOL pre was split, post was
155      |                       split);
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
:= ..... |OP := (FRANGE VAR left, FRANGE CONST right):
          |  CONCR (left) := CONCR (right)
          |ENDOP := ;

:= ..... |OP := (FILE VAR left, FILE CONST right):
          |  EXTERNAL 260
          |END OP :=;

becomes ..... |PROC becomes (INT VAR a, b) :
              |  INTERNAL 260 ;
              |  a := b
              |END PROC becomes;

initialize ..... |PROC initialize (FILE VAR f) :
                |
                |  f.used      := SEQUENCE : (used root, used root, used root, 1, 0);
                |  f.prefix lines := 0;
                |  f.postfix lines := 0;
                |  f.free       := SEQUENCE : (free root, free root, free root, 1, 0);
                |  f.scratch    := SEQUENCE : (scratch root, scratch root, scratch
                |                               root, 1, 0);
                |  f.unused tail := first unused;

```

```

180      | f.limit := 77;
181      | f.edit info := 0;
182      | f.col := 1 ;
183      | f.mark line := 0 ;
184      | f.mark col := 0 ;
185
186      | INT VAR i;
187      | FOR i FROM 1 UPTO 3 REP
188      |   root (i).seg := SEGMENT : (i, i, i);
189      |   root (i).line := ""
190      | PER;
191      | put tabs (f, "") .
192
193      root      | root : f.atoms .
194
195      | END PROC initialize;
196
197
198      | (*****
199      |   **)
200      | (*
201      |   *
202      |   *      Segment Handler      (SEGMENTS & CHAINS)
203      |   *
204      |   *)
205      | (*****
206      |   **)
207
208      segs ..... | INT PROC segs (SEQUENCE CONST s, ATOMROW CONST atom) :
209      |
210      |   INT VAR number of segments := 0 ,
211      |   actual segment := s.segment begin ;
212      |   REP
213      |     number of segments INCR 1 ;
214      |     actual segment := atom (actual segment).seg.succ
215      |   UNTIL actual segment = s.segment begin PER ;
216      |   number of segments .
217
218      | ENDPROC segs ;
219
220
221      nextsegment ..... | PROC next segment (SEQUENCE VAR s, ATOMROW CONST atom) :
222      |
223      |   disable stop;
224      |   s.line no INCR (s.segment end - s.index + 1);
225      |   INT CONST new segment index := actual segment.succ;
226      |   s.segment begin := new segment index;
227      |   s.segment end   := new segment.end;
228      |   s.index         := new segment index .
229
230      |
231
232      actualsegment      | actual segment : atom (s.segment begin).seg .
233
234      newsegment         | new segment :   atom (new segment index).seg .
235
236      | END PROC next segment;

```

```

230
231

232 previoussegment .....|PROC previous segment (SEQUENCE VAR s, ATOMROW CONST atom) :
233
234         disable stop;
235         s.line no DECR (s.index - s.segment begin + 1);
236         INT CONST new segment index := actual segment.pred;
237         s.segment begin := new segment index;
238         s.segment end   := new segment.end;
239         s.index        := s.segment end .
240

241 actualsegment         |actual segment : atom (s.segment begin).seg .
242
243 newsegment            |new segment :   atom (new segment index).seg .
244
245                     |END PROC previous segment;
246

247 splitsegment .....|PROC split segment (SEQUENCE VAR s, ATOMROW VAR atom) :
248
249         disable stop;
250         IF not at segment top
251             THEN split segment at actual position
252         FI .
253

254 splitsegmentatactualpo |split segment at actual position :
255         INT CONST pred index := s.segment begin,
256                 actual index := s.index,
257                 succ index  := pred.succ;
258
259         actual.pred := pred index;
260         actual.succ := succ index;
261         actual.end  := s.segment end;
262
263         pred.succ  := actual index;
264         pred.end   := actual index - 1;
265
266         succ.pred  := actual index;
267
268         s.segment begin := actual index .
269

270 notatsegmenttop       |not at segment top : s.index > s.segment begin .
271

272 pred                  |pred : atom (pred index).seg .
273

274 actual                |actual : atom (actual index).seg .
275

276 succ                  |succ : atom (succ index).seg .
277
278                     |END PROC split segment;
279
280

```

```

281 joinsegments .....|PROC join segments (ATOMROW VAR atom,
282                    |                    INT CONST first index, INT VAR second index) :
283
284                    |    disable stop;
285                    |    IF first seg.end + 1 = second index
286                    |    THEN attach second to first segment
287                    |    ELSE link first to second segment
288                    |    FI .
289
290 attachsecondtofirstseg|attach second to first segment :
291                    |    first seg.end := second seg.end;
292                    |    INT VAR successor of second := second seg.succ;
293                    |    IF successor of second = second index
294                    |    THEN first seg.succ := first index
295                    |    ELSE join segments (atom, first index, successor of second)
296                    |    FI;
297                    |    second index := first index .
298
299 linkfirsttosecondsegme|link first to second segment :
300                    |    first seg.succ := second index;
301                    |    second seg.pred := first index .
302
303 firstseg              |first seg : atom (first index).seg .
304 secondseg             |second seg : atom (second index).seg .
305
306                    |END PROC join segments;
307
308
309 deletesegments .....|PROC delete segments (SEQUENCE VAR from, ATOMROW VAR atom,
310                    |                    INT CONST first index, last index, lines) :
311
312                    |    determine surrounding segments and new atom index;
313                    |    join surrounding segments;
314                    |    update sequence descriptor .
315
316 determinesurroundingse|determine surrounding segments and new atom index :
317                    |    INT VAR pred index := first seg.pred,
318                    |    actual index := last seg.succ;
319                    |    from.index := actual index .
320
321 joinsurroundingsegment|join surrounding segments :
322                    |    join segments (atom, pred index, actual index) .
323
324 updatesequencedescript|update sequence descriptor :
325                    |    from.segment begin := actual index;
326                    |    from.segment end := actual seg.end;
327                    |    from.lines DECR lines .
328
329 actualseg             |actual seg : atom (actual index).seg .

```

```

330     firstseg           |first seg :  atom (first index).seg .
331     lastseg           |last seg :   atom (last index).seg .
332
333     |END PROC delete segments;
334
335
336 insertsegments ..... |PROC insert segments (SEQUENCE VAR into, ATOMROW VAR atom,
337                       |                    INT CONST first index, last index, lines) :
338
339                       |   join into sequence and new segments;
340                       |   update sequence descriptor .
341
342     joinintosequenceandnew |join into sequence and new segments :
343                       |   INT VAR actual index := into.index,
344                       |   pred index := actual seg.pred;
345                       |   join segments (atom, last index, actual index);
346                       |   actual index := first index;
347                       |   join segments (atom, pred index, actual index) .
348
349     updatesequencedescript |update sequence descriptor :
350                       |   into.index := first index;
351                       |   into.segment begin := actual index;
352                       |   into.segment end := actual seg.end;
353                       |   into.lines INCR lines .
354
355     actualseg          |actual seg :  atom (actual index).seg .
356
357     |END PROC insert segments;
358
359
360 nextatom ..... |PROC next atom (SEQUENCE VAR s, ATOMROW CONST atom) :
361
362     |   IF s.line no <= s.lines
363     |   THEN to next atom
364     |   ELSE errorstop ("down" nach Dateiende")
365     |   FI .
366
367     tonextatom        |to next atom :
368     |   disable stop;
369     |   IF s.index = s.segment end
370     |   THEN next segment (s, atom)
371     |   ELSE s.index INCR 1;
372     |         s.line no INCR 1
373     |   FI
374
375     |END PROC next atom;
376
377
378 nextatoms ..... |PROC next atoms (SEQUENCE VAR s, ATOMROW CONST atom, INT CONST
+                       |   times) :
379

```

```

380          | INT CONST destination line := min (s.line no + times, s.lines + 1)
381          | jump upto destination segment;
382          | position within destination segment .
383
384  jumpuptodestinationseg | jump upto destination segment :
385          | WHILE s.line no + length of actual segments tail < destination
+          |   line REP
386          |   next segment (s, atom);
387          | PER .
388
389  positionwithindestinat | position within destination segment :
390          | disable stop;
391          | s.index INCR (destination line - s.line no);
392          | s.line no := destination line .
393
394  lengthofactualegments | length of actual segments tail :  s.segment end - s.index .
395
396          | END PROC next atoms;
397
398
399  previousatom ..... | PROC previous atom (SEQUENCE VAR s, ATOMROW CONST atom) :
400
401          | IF s.line no > 1
402          |   THEN to previous atom
403          |   ELSE errorstop ("up' am Dateianfang")
404          |   FI .
405
406  topreviousatom      | to previous atom :
407          | disable stop;
408          | IF s.index = s.segment begin
409          |   THEN previous segment (s, atom)
410          |   ELSE s.index DECR 1;
411          |   s.line no DECR 1
412          |   FI
413
414          | END PROC previous atom;
415
416
417  previousatoms ..... | PROC previous atoms (SEQUENCE VAR s, ATOMROW CONST atom, INT CONST
+          |   times) :
418
419          | INT CONST destination line := max (1, s.line no - times);
420          | jump back to destination segment;
421          | position within destination segment .
422
423  jumpbacktodestinat | jump back to destination segment :
424          | WHILE s.line no - length of actual segments head > destination
+          |   line REP
425          |   previous segment (s, atom);
426          | PER .
427

```

```

428 positionwithindestinat | position within destination segment :
429 | disable stop;
430 | s.index DECR (s.line no - destination line);
431 | s.line no := destination line .
432 |
433 lengthofactualsegments | length of actual segments head : s.index - s.segment begin .
434 |
435 | END PROC previous atoms;
436 |
437 |
438 | TEXT VAR pre, pat, pattern@;
439 | INT VAR last search line ;
440 |
441 searchdown ..... | PROC search down (SEQUENCE VAR s, ATOMROW CONST atom, TEXT CONST
+ | pattern,
442 | INT CONST max lines, INT VAR column) :
443 |
444 | INT CONST start col := column ,
445 | start line := s.lineno ;
446 | last search line := min (s.lines, s.lineno + max lines) ;
447 | pre:= somefix (pattern) ;
448 | pattern@ := pattern ** @ ;
449 | down in atoms (s, atom, pre, column);
450 | IF NOT (last search succeeded CAND like pattern)
451 | THEN try again
452 | FI;
453 | last search succeeded := TRUE ;
454 | column := matchpos ( @ ) .
455 |
456 tryagain | try again:
457 | WHILE s.line no < last search line
458 | REP next atom (s, atom) ;
459 | column := 1 ;
460 | down in atoms (s, atom, pre, column);
461 | IF last search succeeded CAND like pattern
462 | THEN LEAVE try again
463 | FI
464 | PER;
465 | column := 1 + LENGTH record;
466 | last search succeeded := FALSE ;
467 | LEAVE search down.
468 |
469 likepattern | like pattern :
470 | correct position ;
471 | pat := any (column-1) ;
472 | pat CAT any ;
473 | pat CAT pattern@ ;
474 | pat CAT any ;
475 | record LIKE pat .
476 |
477 correctposition | correct position :
478 | IF s.lineno = start line
479 | THEN column := start col
480 | ELSE column := 1
481 | FI .
482 |

```

```

483 record |record : atom (s.index).line .
484
485 |ENDPROC search down ;
486 |
.

487 downinatoms .....|PROC down in atoms (SEQUENCE VAR s, ATOMROW CONST atom, TEXT CONST
+ |
488 | pattern,
489 | INT VAR column) :
490 | last search succeeded := FALSE ;
491 | search forwards in actual line ;
492 | IF NOT found AND s.line no < last search line
493 | THEN search in following lines
494 | FI ;
495 | IF found
496 | THEN last search succeeded := TRUE
497 | ELSE set column behind last char
498 | FI .
499 |

500 setcolumnbehindlastcha |set column behind last char :
501 | column := LENGTH atom (s.index).line + 1 .
502 |

503 searchforwardsinactual |search forwards in actual line :
504 | IF pattern <> ""
505 | THEN column := pos (atom (s.index).line, pattern, column)
506 | ELIF column > LENGTH atom (s.index).line
507 | THEN column := @
508 | FI .
509 |

510 searchinfollowinglines |search in following lines :
511 | next atom (s, atom) ;
512 | IF pattern = ""
513 | THEN column := 1 ;
514 | LEAVE search in following lines
515 | FI ;
516 | REP
517 | search forwards through segment ;
518 | update file position forwards ;
519 | IF found OR s.line no = last search line
520 | THEN LEAVE search in following lines
521 | ELSE next segment (s, atom)
522 | FI
523 | PER .
524 |

525 searchforwardsthroughs |search forwards through segment :
526 | INT VAR search index := s.index ,
527 | last index := min (s.segment end, s.index+(last search line-s.line
+ | no));
528 | REP
529 | column := pos (atom (search index).line, pattern) ;
530 | IF found OR search index = last index
531 | THEN LEAVE search forwards through segment
532 | FI ;
533 | search index INCR 1
534 | PER .
535 |

```

```

536 updatefilepositionforw | update file position forwards :
537 | disable stop ;
538 | s.line no INCR (search index - s.index) ;
539 | s.index := search index ;
540 | enable stop .
541 |
542 found | found : column > 0 .
543 |
544 | ENDPROC down in atoms ;
545 |

546 prefix ..... | TEXT PROC prefix (TEXT CONST pattern) :
547 |
548 | INT VAR invalid char pos := pos (pattern, ""0"", ""31"", 1) ;
549 | SELECT invalid char pos OF
550 | CASE 0 : pattern
551 | CASE 1 : ""
552 | OTHERWISE : subtext (pattern, 1, invalid char pos - 1)
553 | ENDSELECT .
554 |
555 | ENDPROC prefix ;
556 |

557 searchup ..... | PROC search up (SEQUENCE VAR s, ATOMROW CONST atom, TEXT CONST
+ | pattern,
558 | INT CONST max lines, INT VAR column) :
559 |
560 | last search line := max (1, s.lineno - max lines) ;
561 | pre:= prefix (pattern);
562 | pattern0 := pattern ** 0;
563 | remember start point ;
564 | up in atoms (s, atom, pre, column);
565 | IF NOT (last search succeeded CAND last pattern in line found)
566 | THEN try again
567 | FI;
568 | last search succeeded := TRUE ;
569 | column := matchpos (0) .
570 |

571 tryagain | try again:
572 | WHILE s.lineno > last search line OR column > 1
573 | REP previous atom (s, atom);
574 | column := LENGTH record ;
575 | up in atoms (s, atom, pre, column);
576 | IF last search succeeded CAND last pattern in line found
577 | THEN LEAVE try again
578 | FI
579 | PER;
580 | column := 1;
581 | last search succeeded := FALSE ;
582 | LEAVE search up.
583 |

584 rememberstartpoint | remember start point :
585 | INT VAR c:= column, r:= s.lineno;.
586 |

```

```

587 lastpatterninlinefound | last pattern in line found :
588                       | column := 2 ;
589                       | WHILE like pattern CAND right of start REP
590                       |   column := matchpos (0) +1
591                       | PER ;
592                       | column DECR 1 ;
593                       | like pattern CAND right of start .
594
595 likepattern            | like pattern :
596                       | pat := any (column-1) ;
597                       | pat CAT any ;
598                       | pat CAT pattern@ ;
599                       | pat CAT any ;
600                       | record LIKE pat .
601
602 rightofstart          | right of start : (r > s.line no COR c >= matchpos(0)) .
603
604 record                | record          : atom (s.index).line .
605
606                       | ENDPROC search up ;
607
607 upinatoms ..... | PROC up in atoms (SEQUENCE VAR s, ATOMROW CONST atom, TEXT CONST
+                       |   pattern,
608                       |   INT VAR column) :
609
610                       | last search succeeded := FALSE ;
611                       | search backwards in actual line ;
612                       | IF NOT found AND s.line no > last search line
613                       |   THEN search in preceding lines
614                       | FI ;
615                       | IF found
616                       |   THEN last search succeeded := TRUE
617                       |   ELSE column := 1
618                       | FI .
619
620 searchbackwardsinactua | search backwards in actual line :
621                       | IF pattern = ""
622                       |   THEN LEAVE search backwards in actual line
623                       | FI ;
624                       | INT VAR last pos , new pos := 0 ;
625                       | REP
626                       |   last pos := new pos ;
627                       |   new pos := pos (atom (s.index).line, pattern, last pos+1) ;
628                       | UNTIL new pos = 0 OR new pos > column PER ;
629                       | column := last pos .
630
631 searchinprecedingline | search in preceding lines :
632                       | previous atom (s, atom) ;
633                       | IF pattern = ""
634                       |   THEN column := LENGTH atom (s.index).line + 1 ;
635                       |   last search succeeded := TRUE ;
636                       |   LEAVE search in preceding lines
637                       | FI ;
638                       | REP
639                       |   search backwards through segment ;
640                       |   update file position backwards ;

```

```

641                                     |   IF found OR s.line no = last search line
642                                     |   THEN LEAVE search in preceding lines
643                                     |   ELSE previous segment (s, atom)
644                                     |   FI
645                                     |   PER .
646
647 searchbackwardsthrough |search backwards through segment :
648                                     | INT VAR search index := s.index ,
649                                     | last index := max (s.segment begin, s.index-(s.line no-last search
+                                     | line));
650                                     | REP
651                                     |   new pos := 0 ;
652                                     |   REP
653                                     |     column := new pos ;
654                                     |     new pos := pos (atom (search index).line, pattern, column+1) ;
655                                     |   UNTIL new pos = 0 PER ;
656                                     |   IF found OR search index = last index
657                                     |   THEN LEAVE search backwards through segment
658                                     |   FI ;
659                                     |   search index DECR 1
660                                     |   PER .
661
662 updatefilepositionback |update file position backwards :
663                                     | disable stop ;
664                                     | s.line no DECR (s.index - search index) ;
665                                     | s.index := search index ;
666                                     | enable stop .
667
668 found                               |found : column > 0 .
669
670                                     |ENDPROC up in atoms ;
671
672                                     |BOOL VAR last search succeeded ;
673
674 patternfound ..... |BOOL PROC pattern found :
675                                     | last search succeeded
676                                     |ENDPROC pattern found ;
677
678
679
680 deleteatom ..... |PROC delete atom (SEQUENCE VAR used, free, ATOMROW VAR atom) :
681                                     |
682                                     | disable stop;
683                                     | IF used.line no <= used.lines
684                                     | THEN delete actual atom
685                                     | ELSE errorstop ("delete' am Dateiende")
686                                     | FI .
687
688 deleteactualatom |delete actual atom :
689                                     | position behind actual free segment;
690                                     | split segment (used, atom);
691                                     | INT VAR actual index := used.index;
692                                     | cut off tail of actual used segment;
693                                     | delete segments (used, atom, actual index, actual index, 1);

```

```

694      insert segments (free, atom, actual index, actual index, 1) .
695
696      positionbehindactualfr position behind actual free segment :
697      IF free.line no <= free.lines
698      THEN next segment (free, atom)
699      FI .
700
701      cutofftailofactualused cut off tail of actual used segment :
702      IF actual index <> used.segment end
703      THEN used.index INCR 1;
704      split segment (used, atom);
705      used.index DECR 1
706      FI .
707
708      END PROC delete atom;
709
710
711      insertatom ..... PROC insert atom (SEQUENCE VAR used, free, INT VAR unused, ATOMROW
+      VAR atom) :
712
713      disable stop;
714      split segment (used, atom);
715      IF free.lines > 0
716      THEN insert new atom from free sequence
717      ELIF unused <= file size
718      THEN insert new atom from unused tail
719      ELSE errorstop ("FILE-Ueberlauf")
720      FI .
721
722      insertnewatomfromfrees insert new atom from free sequence :
723      get a free segments head;
724      make this atom to actual segment;
725      transfer from free to used chain .
726
727      getafreesegmentshead get a free segments head :
728      IF actual free segment is root segment
729      THEN previous segment (free, atom)
730      FI;
731      position to actual segments head .
732
733      positiontoactualsegmen position to actual segments head :
734      INT VAR actual index := free.segment begin;
735      free.line no DECR (free.index - actual index);
736      free.index := actual index .
737
738      makethisatomtoactuale make this atom to actual segment :
739      IF free.segment end > actual index
740      THEN free.index INCR 1;
741      split segment (free, atom);
742      free.index DECR 1
743      FI .
744

```

```

745 transferfromfreetoused |transfer from free to used chain :
746 |delete segments (free, atom, actual index, actual index, 1);
747 |insert segments (used, atom, actual index, actual index, 1);
748 |atom (actual index).line := "" .
749 |
750 insertnewatomfromunuse |insert new atom from unused tail :
751 |actual index := unused;
752 |atom (actual index).seg :=
753 |         SEGMENT:(actual index, actual index, actual
754 |         +         index);
755 |atom (actual index).line := "";
756 |insert segments (used, atom, actual index, actual index, 1);
757 |unused INCR 1 .
758 actualfreesegmentisroo |actual free segment is root segment :   free.segment begin = free
759 |         root .
760 |END PROC insert atom;
761 |
762 |
763 insertnext ..... |PROC insert next (SEQUENCE VAR used, free, INT VAR unused, ATOMROW
764 |         VAR atom,
765 |         TEXT CONST record) :
766 |
767 |         IF used.line no > used.lines
768 |         THEN insert atom (used, free, unused, atom)
769 |         ELIF actual position before unused nonempty atomrow part
770 |         THEN forward and insert atom by simple extension of used atomrow
771 |         part
772 |         ELSE next atom (used, atom);
773 |         insert atom (used, free, unused, atom)
774 |         FI;
775 |         atom (used.index).line := record .
776 |
777 forwardandinsertatomby |forward and insert atom by simple extension of used atomrow part :
778 |used.line no INCR 1;
779 |used.lines INCR 1;
780 |used.index INCR 1;
781 |used.segment end INCR 1;
782 |atom (used.segment begin).seg.end INCR 1;
783 |unused INCR 1 .
784 |
785 actualpositionbeforeun |actual position before unused nonempty atomrow part :
786 |used.index = unused - 1 AND unused part not empty .
787 |
788 unusedpartnotempty |unused part not empty : unused <= file size .
789 |
790 |
791 transfersubsequence ..... |PROC transfer subsequence (SEQUENCE VAR source, dest,
792 |         ATOMROW VAR atom, INT CONST size) :
793 |

```

```

794      IF size > 0
795          THEN INT VAR subsequence size := min (size, source.line no);
796              mark begin of source part;
797              mark end of source part;
798              split destination sequence;
799              transfer part
800
801      FI .

802 markbeginofsourcepart mark begin of source part :
803     previous atoms (source, atom, subsequence size - 1);
804     split segment (source, atom);
805     INT CONST first := source.segment begin .
806

807 markendofsourcepart mark end of source part :
808     next atoms (source, atom, subsequence size - 1);
809     INT CONST last := source.segment begin;
810     next atom (source, atom);
811     split segment (source, atom) .
812

813 splitdestinationsequen split destination sequence :
814     split segment (dest, atom) .
815

816 transferpart transfer part :
817     disable stop;
818     delete segments (source, atom, first, last, subsequence size);
819     source.line no DECR subsequence size;
820     insert segments (dest, atom, first, last, subsequence size);
821     next atoms (dest, atom, subsequence size - 1) .
822
823 END PROC transfer subsequence;
824
825
826
827 (*****
+
828 (****
+
829 (**** FILE handler
+
830 (****
+
831 (*****
+
832 )
833
834
835 LET file type = 1003 ,
836     file type 16 = 1002 ,
837
838     closed = 0,
839     inp = 1,
840     outp = 2,
841     mod = 3,
842     end = 4,
843
844     max limit = 16000,
845     super limit = 16001;
846

```

```

847
848 |TYPE TRANSPUTDIRECTION = INT;
849 |
850 |

851 input .....|TRANSPUTDIRECTION PROC input :
852 |   TRANSPUTDIRECTION : (inp)
853 |   END PROC input;
854 |
855 |

856 output .....|TRANSPUTDIRECTION PROC output :
857 |   TRANSPUTDIRECTION : (outp)
858 |   END PROC output;
859 |
860 |

861 modify .....|TRANSPUTDIRECTION PROC modify :
862 |   TRANSPUTDIRECTION : (mod)
863 |   END PROC modify;
864 |
865 |
866 |   FILE VAR result file;
867 |
868 |

869 sequentialfile .....|FILE PROC sequential file (TRANSPUTDIRECTION CONST mode,
870 |                               DATASPACE CONST ds) :
871 |   IF type (ds) = file type
872 |   THEN result := ds
873 |   ELIF type (ds) < 0
874 |   THEN result := ds; type (ds, file type); initialize (result file)
875 |   ELSE enable stop; errorstop ("Datenraum hat falschen Typ")
876 |   FI;
877 |   reset (result file, mode);
878 |   result file .
879 |

880 result |result : CONCR (result file) .
881 |
882 |   END PROC sequential file;
883 |
884 |

885 sequentialfile .....|FILE PROC sequential file (TRANSPUTDIRECTION CONST mode, TEXT CONST
+ |   name) :
886 |
887 |   IF exists (name)
888 |   THEN get dataspace if file
889 |   ELIF CONCR (mode) <> inp
890 |   THEN get new file space
891 |   ELSE errorstop ("""+name+"" gibt es nicht") ; enable stop
892 |   FI;
893 |   update status if necessary;
894 |   reset (result file, mode);
895 |   result file .
896 |

```

```

897  getdataspaceiffile |get dataspace if file :
898                      | IF type (old (name)) = file type 16
899                      | THEN reorganize (name)
900                      | FI ;
901                      | result := old (name, file type) ;
902                      | IF is 17@ file
903                      | THEN result.col := 1 ;
904                      |      result.mark line := @ ;
905                      |      result.mark col := @
906                      | FI .
907
908  is17@file            |is 17@ file : result.mark col < @ .
909
910  getnewfilespace     |get new file space :
911                      | result := new (name);
912                      | IF NOT is error
913                      | THEN type (old (name), file type); initialize (result file)
914                      | FI .
915
916  updatetestatusifnecessar |update status if necessary :
917                      | IF CONCR (mode) <> inp
918                      | THEN status (name, ""); headline (result file, name)
919                      | FI .
920
921  result              |result : CONCR (result file) .
922
923                      |END PROC sequential file;
924
925
926  reset .....        |PROC reset (FILE VAR f) :
927                      |
928                      | IF f.mode = end
929                      | THEN reset (f, input)
930                      | ELSE reset (f, TRANSPUTDIRECTION:(f.mode))
931                      | FI .
932
933                      |ENDPROC reset ;
934
935  reset .....        |PROC reset (FILE VAR f, TRANSPUTDIRECTION CONST mode) :
936                      |
937                      | IF f.mode <> mod OR new mode <> mod
938                      | THEN f.mode := new mode ;
939                      |      initialize file index
940                      | FI .
941
942  initializefileindex |initialize file index :
943                      | IF new mode = outp
944                      | THEN to line without check (f, f.used.lines);
945                      |      col := super limit
946                      | ELSE to line without check (f, 1);
947                      |      col := 1 ;
948                      | IF new mode = inp AND file is empty
949                      | THEN f.mode := end

```

```

950                                     |         FI
951                                     |         FI .
952                                     |
953     fileisempty                       | file is empty : f.used.lines = 0 .
954                                     |
955     newmode                            | new mode : CONCR (mode) .
956                                     |
957     col                                | col : CONCR (CONCR (f)).col .
958                                     |
959                                     | END PROC reset;
960                                     |
961                                     |
962     input .....                       | PROC input (FILE VAR f) :
963                                     |
964                                     |     reset (f, input) .
965                                     |
966                                     | END PROC input;
967                                     |
968                                     |
969     output .....                       | PROC output (FILE VAR f) :
970                                     |
971                                     |     reset (f, output)
972                                     |
973                                     | END PROC output;
974                                     |
975                                     |
976     modify .....                       | PROC modify (FILE VAR f) :
977                                     |
978                                     |     reset (f, modify)
979                                     |
980                                     | END PROC modify;
981                                     |
982                                     |
983     close .....                        | PROC close (FILE VAR f) :
984                                     |
985                                     |     f.mode := closed .
986                                     |
987                                     | END PROC close;
988                                     |
989                                     |
990     checkmode .....                   | PROC check mode (FILE CONST f, INT CONST mode) :
991                                     |
992                                     |     IF f.mode = mode
993                                     |     THEN LEAVE check mode
994                                     |     ELIF f.mode = closed
995                                     |     THEN errorstop ("Datei zu!")
996                                     |     ELIF f.mode = mod
997                                     |     THEN errorstop ("unzulaessiger Zugriff auf modify-FILE")
998                                     |     ELIF mode = mod

```

```

999          | THEN errorstop ("Zugriff nur auf modify-FILE zulaessig")
1000         | ELIF f.mode = end
1001         | THEN errorstop ("Leseversuch nach Dateiende")
1002         | ELIF mode = inp
1003         | THEN errorstop ("Leseversuch auf output-FILE")
1004         | ELIF mode = outp
1005         | THEN errorstop ("Schreibversuch auf input-FILE")
1006         | FI .
1007
1008         | END PROC check mode;
1009
1010
1011 tolinewithoutcheck .....| PROC to line without check (FILE VAR f, INT CONST destination line)
1012
1013         | INT CONST distance := destination line - f.used.line no;
1014         | IF distance > 0
1015         | THEN next atoms (f.used, f.atoms, distance)
1016         | ELIF distance < 0
1017         | THEN previous atoms (f.used, f.atoms, - distance)
1018         | FI .
1019
1020         | END PROC to line without check;
1021
1022
1023 toline .....| PROC to line (FILE VAR f, INT CONST destination line) :
1024
1025         | check mode (f, mod);
1026         | to line without check (f, destination line)
1027
1028         | END PROC to line;
1029
1030
1031 tofirstrecord .....| PROC to first record (FILE VAR f) :
1032
1033         | to line (f, 1)
1034
1035         | END PROC to first record;
1036
1037
1038 toeof .....| PROC to eof (FILE VAR f) :
1039
1040         | to line (f, f.used.lines + 1) .
1041
1042         | END PROC to eof;
1043
1044
1045 putline .....| PROC putline (FILE VAR f, TEXT CONST word) :
1046
1047         | write (f, word);
1048         | col := super limit .
1049

```

```

1050 col |col : CONCR (CONCR (f)).col .
1051
1052 |END PROC putline;
1053
1054 |
|
|
1055 deleterecord .....|PROC delete record (FILE VAR f) :
1056 |
1057 |   check mode (f, mod);
1058 |   delete atom (f.used, f.free, f.atoms) .
1059 |
1060 |END PROC delete record;
1061
1062 |
|
|
1063 insertrecord .....|PROC insert record (FILE VAR f) :
1064 |
1065 |   check mode (f, mod);
1066 |   insert atom (f.used, f.free, f.unused tail, f.atoms) .
1067 |
1068 |END PROC insert record;
1069
1070 |
|
|
1071 down .....|PROC down (FILE VAR f) :
1072 |
1073 |   check mode (f, mod);
1074 |   next atom (f.used, f.atoms) .
1075 |
1076 |END PROC down ;
1077
|
|
|
1078 up .....|PROC up (FILE VAR f) :
1079 |
1080 |   check mode (f, mod);
1081 |   previous atom (f.used, f.atoms) .
1082 |
1083 |END PROC up ;
1084
|
|
|
1085 down .....|PROC down (FILE VAR f, INT CONST n) :
1086 |
1087 |   to line (f, lineno (f) + n)
1088 |
1089 |ENDPROC down ;
1090
|
|
|
1091 up .....|PROC up (FILE VAR f, INT CONST n) :
1092 |
1093 |   to line (f, lineno (f) - n)
1094 |
1095 |ENDPROC up ;
1096
1097 |

```

```

1098 writerecord .....|PROC write record (FILE VAR f, TEXT CONST record) :
1099                    |
1100                    |   check mode (f, mod);
1101                    |   IF   not at eof
1102                    |       THEN f.atoms (f.used.index).line := record
1103                    |            ELSE errorstop ("write' nach Dateiende")
1104                    |            FI .
1105                    |
1106 notateof            |not at eof : f.used.line no <= f.used.lines .
1107                    |
1108                    |END PROC write record;
1109                    |
1110                    |
1111 readrecord .....|PROC read record (FILE CONST f, TEXT VAR record) :
1112                    |
1113                    |   check mode (f, mod);
1114                    |   record := f.atoms (f.used.index).line .
1115                    |
1116                    |END PROC read record;
1117                    |
1118                    |
1119 line .....|PROC line (FILE VAR f) :
1120                    |
1121                    |   IF   mode = end
1122                    |       THEN errorstop ("Leseversuch nach Dateiende")
1123                    |       ELIF mode = inp
1124                    |           THEN next atom (f.used, f.atoms); col := 1; check eof
1125                    |       ELIF mode = outp
1126                    |           THEN IF col <= max limit
1127                    |                   THEN col := super limit
1128                    |                   ELSE append empty line
1129                    |           FI
1130                    |   FI .
1131                    |
1132 appendemptyline    |append empty line :
1133                    |   insert next (f.used, f.free, f.unused tail, f.atoms, "") .
1134                    |
1135 col                |col : CONCR (CONCR (f)).col .
1136                    |
1137 mode              |mode : CONCR (CONCR (f)).mode .
1138                    |
1139 checkeof          |check eof :
1140                    |   IF eof (f) THEN mode := end FI .
1141                    |
1142                    |END PROC line;
1143                    |
1144                    |
1145 line .....|PROC line (FILE VAR f, INT CONST lines) :
1146                    |
1147                    |   INT VAR i; FOR i FROM 1 UPTO lines REP line (f) PER
1148                    |

```

```

1149                                     |END PROC line;
1150                                     |
1151                                     |

1152  getline .....|PROC getline (FILE VAR f, TEXT VAR text) :
1153                                     |
1154                                     |  check mode (f, inp);
1155                                     |  text := subtext (record, f.col);
1156                                     |  IF f.used.line no >= f.used.lines
1157                                     |      THEN f.mode := end ;
1158                                     |           set end of file
1159                                     |      ELSE to next line ;
1160                                     |           f.col := 1
1161                                     |  FI .
1162                                     |

1163  tonextline   |to next line :
1164                                     |  next atom (f.used, f.atoms) .
1165                                     |

1166  setendoffile |set end of file :
1167                                     |  f.col := LENGTH record + 1 .
1168                                     |

1169  record       |record : f.atoms (f.used.index).line .
1170                                     |
1171                                     |END PROC getline;
1172                                     |
1173                                     |

1174  isfirstrecord .....|BOOL PROC is first record (FILE CONST f) :
1175                                     |
1176                                     |  check mode (f, mod);
1177                                     |  f.used.line no = 1 .
1178                                     |
1179                                     |END PROC is first record;
1180                                     |
1181                                     |

1182  eof .....|BOOL PROC eof (FILE CONST f) :
1183                                     |
1184                                     |  IF line no < lines THEN FALSE
1185                                     |  ELIF line no = lines THEN col > LENGTH record
1186                                     |  ELSE TRUE
1187                                     |  FI .
1188                                     |

1189  lineno       |line no : f.used.line no .
1190                                     |
1191  lines        |lines  : f.used.lines .
1192                                     |
1193  col          |col    : f.col .
1194                                     |
1195  record       |record : f.atoms (f.used.index).line .
1196                                     |
1197                                     |END PROC eof;
1198                                     |
1199                                     |

```

```

1197 lineno .....|INT PROC line no (FILE CONST f) :
1198                |
1199                |   f.used.line no .
1200                |
1201                |END PROC line no;
1202                |
1203                |

1204 linetype .....|PROC line type (FILE VAR f, INT CONST t) :
1205                |
1206                |   f.atoms (f.used.index).type := t .
1207                |
1208                |ENDPROC line type ;
1209                |

1210 linetype .....|INT PROC line type (FILE CONST f) :
1211                |
1212                |   f.atoms (f.used.index).type .
1213                |
1214                |ENDPROC line type ;
1215                |
1216                |

1217 put .....|PROC put (FILE VAR f, TEXT CONST word) :
1218                |
1219                |   check mode (f, outp);
1220                |   IF col + LENGTH word > f.limit
1221                |       THEN append new line
1222                |           ELSE record CAT word
1223                |   FI;
1224                |   record CAT " ";
1225                |   col := LENGTH record + 1 .
1226                |

1227     appendnewline      |append new line :
1228                        |   insert next (f.used, f.free, f.unused tail, f.atoms, word) .
1229                        |

1230     record              |record : f.atoms (f.used.index).line .
1231
1232     col                  |col :   f.col .
1233
1234                        |END PROC put;
1235                        |

1236 put .....|PROC put (FILE VAR f, INT CONST value) :
1237                |
1238                |   put (f, text (value))
1239                |
1240                |END PROC put;
1241                |
1242                |

1243 put .....|PROC put (FILE VAR f, REAL CONST real) :
1244                |
1245                |   put (f, text (real))

```

```

1246
1247 |END PROC put;
1248
1249

1250 write .....|PROC write (FILE VAR f, TEXT CONST word) :
1251
1252 |   check mode (f, outp);
1253 |   IF col + LENGTH word - 1 > f.limit
1254 |     THEN append new line
1255 |     ELSE record CAT word
1256 |   FI;
1257 |   col := LENGTH record + 1 .
1258

1259   appendnewline |append new line :
1260 |   insert next (f.used, f.free, f.unused tail, f.atoms, word) .
1261

1262   record |record : f.atoms (f.used.index).line .

1263   col |col : f.col .
1264
1265 |END PROC write;
1266
1267

1268 get .....|PROC get (FILE VAR f, TEXT VAR word, TEXT CONST separator) :
1269
1270 |   check mode (f, inp);
1271 |   skip separators;
1272 |   IF word found
1273 |     THEN get word
1274 |     ELSE try to find word in next line
1275 |   FI .
1276

1277   skipseparators |skip separators :
1278 |   INT CONST separator length := LENGTH separator;
1279 |   WHILE is separator REP col INCR separator length PER .

1281   isseparator |is separator :
1282 |   subtext (record, col, col + separator length - 1) = separator .
1283

1284   wordfound |word found : col <= LENGTH record .

1286   getword |get word :
1287 |   INT VAR end of word := pos (record, separator, col) - 1;
1288 |   IF separator found
1289 |     THEN get text upto separator
1290 |     ELSE get rest of record
1291 |   FI .

1293   separatorfound |separator found : end of word >= 0 .
1294

```

```

1295  gettextuptoseparator |get text upto separator :
1296                        | word := subtext (record, col, end of word);
1297                        | col := end of word + separator length + 1;
1298                        | IF col > LENGTH record THEN line (f) FI .
1299
1300  getrestofrecord       |get rest of record :
1301                        | word := subtext (record, col); line (f) .
1302
1303  record                |record : f.atoms (f.used.index).line .
1304
1304  col                   |col   : f.col .
1305
1306  trytofindwordinnextlin |try to find word in next line :
1307                        | line (f); IF eof (f) THEN word := "" ELSE get (f, word, separator)
1308                        | FI .
1309                        |
1310                        | END PROC get;
1311

```

```

1312  get ..... |PROC get (FILE VAR f, TEXT VAR word, INT CONST max length) :
1313
1314                        | check mode (f, inp);
1315                        | IF word is only a part of record
1316                        | THEN get text of certain length
1317                        | ELSE get rest of record
1318                        | FI .
1319

```

```

1320  wordisonlyapartofrecor |word is only a part of record :
1321                        | col <= LENGTH record - max length .
1322

```

```

1323  gettextofcertainlength |get text of certain length :
1324                        | word := text (record, max length, col);
1325                        | col INCR max length .
1326

```

```

1327  getrestofrecord       |get rest of record :
1328                        | word := subtext (record, col); line (f) .
1329

```

```

1330  record                |record : f.atoms (f.used.index).line .

```

```

1331  col                   |col   : f.col .

```

```

1333                        | END PROC get;

```

```

1336  get ..... |PROC get (FILE VAR f, TEXT VAR word) :

```

```

1337                        | get (f, word, " ")

```

```

1340                        | END PROC get;

```

```
1343 |TEXT VAR number word;
1344 |
1345 |
```

```
1346 get .....|PROC get (FILE VAR f, INT VAR number) :
1347 |
1348 |   get (f, number word);
1349 |   number := int (number word)
1350 |
1351 |END PROC get;
1352 |
1353 |
```

```
1354 get .....|PROC get (FILE VAR f, REAL VAR number) :
1355 |
1356 |   get (f, number word);
1357 |   number := real (number word)
1358 |
1359 |END PROC get;
1360 |
1361 |TEXT VAR split record ;
1362 |INT VAR indentation ;
1363 |
1364 |
```

```
1365 splitline .....|PROC split line (FILE VAR f, INT CONST split col) :
1366 |
1367 |   split line (f, split col, TRUE)
1368 |
1369 |ENDPROC split line ;
1370 |
```

```
1371 splitline .....|PROC split line (FILE VAR f, INT CONST split col, BOOL CONST note
+ |   indentation) :
1372 |
1373 |   IF note indentation
1374 |   THEN get indentation
1375 |   ELSE indentation := @
1376 |   FI ;
1377 |   get split record ;
1378 |   insert split record and indentation ;
1379 |   cut off old record .
1380 |
```

```
1381 getindentation |get indentation :
1382 |   indentation := pos (actual record, ""33"", ""254"", 1) - 1 ;
1383 |   IF indentation < @ OR indentation >= split col
1384 |   THEN indentation := split col - 1
1385 |   FI .
1386 |
```

```
1387 getsplitrecord |get split record :
1388 |   split record := subtext (actual record, split col, max limit)
1389 |
```

```
1390 insertsplitrecordandin |insert split record and indentation :
1391 |   down (f) ;
1392 |   insert record (f) ;
```

```

1393          | INT VAR i ;
1394          | FOR i FROM 1 UPTO indentation REP
1395          |     actual record CAT " "
1396          | PER ;
1397          | actual record CAT split record ;
1398          | up (f) .
1399          |
1400  cutoffoldrecord | cut off old record :
1401          |     actual record := subtext (actual record, 1, split col-1) .
1402          |
1403  actualrecord    | actual record : f.atoms (f.used.index).line .
1404          |
1405          | ENDPROC split line ;
1406          |
1407  concatenateline .....| PROC concatenate line (FILE VAR f, BOOL CONST delete blanks) :
1408          |
1409          |     down (f) ;
1410          |     split record := actual record ;
1411          |     IF delete blanks
1412          |         THEN delete leading blanks
1413          |     FI ;
1414          |     delete record (f) ;
1415          |     up (f) ;
1416          |     actual record CAT split record .
1417          |
1418  deleteleadingblanks | delete leading blanks :
1419          |     INT CONST non blank col := pos (split record, ""33"", ""254"", 1)
1420          |     IF non blank col > 0
1421          |         THEN split record := subtext (split record, non blank col)
1422          |     FI .
1423          |
1424  actualrecord    | actual record : f.atoms (f.used.index).line .
1425          |
1426          | ENDPROC concatenate line ;
1427          |
1428  concatenateline .....| PROC concatenate line (FILE VAR f) :
1429          |     concatenate line (f, TRUE)
1430          | ENDPROC concatenate line ;
1431          |
1432  reorganize .....| PROC reorganize :
1433          |
1434          |     reorganize (last param)
1435          |
1436          | END PROC reorganize;
1437          |
1438          |
1439          | TEXT VAR file record ;
1440          |

```

```

1441 reorganize .....|PROC reorganize (TEXT CONST file name) :
1442 |
1443 |   enable stop ;
1444 |   FILE VAR input file, output file;
1445 |   DATASPACE VAR scratch space;
1446 |   INT CONST type of dataspace := type (old (file name)) ;
1447 |   INT VAR counter;
1448 |
1449 |   last param (file name);
1450 |   IF type of dataspace = file type
1451 |     THEN reorganize new to new
1452 |   ELIF type of dataspace = file type 16
1453 |     THEN reorganize old to new
1454 |     ELSE errorstop ("Datenraum hat falschen Typ")
1455 |   FI;
1456 |   replace file space by scratch space .
1457 |
1458 reorganizenewtonew |reorganize new to new :
1459 |   input file := sequential file (input, file name);
1460 |   disable stop ;
1461 |   scratch space := nilspace ;
1462 |   output file := sequential file (output, scratch space);
1463 |   copy attributes (input file, output file) ;
1464 |
1465 |   FOR counter FROM 1 UPTO 9999
1466 |   WHILE NOT eof (input file) REP
1467 |     cout (counter);
1468 |     getline (input file, file record);
1469 |     putline (output file, file record);
1470 |     check for interrupt
1471 |   PER .
1472 |
1473 reorganizeoldtonew |reorganize old to new :
1474 |   LET OLDRECORD = STRUCT (INT succ, pred, x, y, TEXT record);
1475 |   LET OLDFILE = BOUND ROW 4075 OLDRECORD;
1476 |   LET dateianker = 2, freianker = 1;
1477 |   INT VAR index := dateianker;
1478 |
1479 |   OLDFILE VAR old file := old (file name);
1480 |   disable stop;
1481 |   scratch space := nilspace;
1482 |   output file := sequential file (output, scratch space);
1483 |   get old attributes ;
1484 |
1485 |   say ("Datei wird in 1.7-Format gewandelt: ") ;
1486 |
1487 |   FOR counter FROM 1 UPTO 9999
1488 |   WHILE NOT end of old file REP
1489 |     cout (counter);
1490 |     index := next record;
1491 |     file record := record of old file ;
1492 |     IF pos (file record, ""128"", ""250"", 1) > 0
1493 |       THEN change special chars
1494 |     FI ;
1495 |     putline (output file, file record);
1496 |     check for interrupt
1497 |   PER .
1498 |

```

```

1499  getoldattributes |get old attributes :
1500                    |  get old headline ;
1501                    |  get old limit and tabs .
1502
1503  getoldheadline     |get old headline :
1504                    |  headline (output file, old file (dateianker).record) .
1505
1506  getoldlimitandtabs |get old limit and tabs :
1507                    |  file record := old file (freianker).record ;
1508                    |  max line length (output file, int (subtext (file record, 11, 15))
1509                    |  ;
1510                    |  put tabs (output file, subtext (file record, 16)) .
1511
1511  changespecialchars |change special chars :
1512                    |  change all (file record, ""193"", ""214"") (* Ae *) ;
1513                    |  change all (file record, ""207"", ""215"") (* Oe *) ;
1514                    |  change all (file record, ""213"", ""216"") (* Ue *) ;
1515                    |  change all (file record, ""225"", ""217"") (* ae *) ;
1516                    |  change all (file record, ""239"", ""218"") (* oe *) ;
1517                    |  change all (file record, ""245"", ""219"") (* ue *) ;
1518                    |  change all (file record, ""235"", ""220"") (* k *) ;
1519                    |  change all (file record, ""173"", ""221"") (* - *) ;
1520                    |  change all (file record, ""163"", ""222"") (* fis *) ;
1521                    |  change all (file record, ""160"", ""223"") (* blank *) ;
1522                    |  change all (file record, ""194"", ""251"") (* eszet *) .
1523
1524  endofoldfile       |end of old file :      next record = dateianker .
1525
1526  nextrecord         |next record :          old file (index).succ .
1527
1528  recordofoldfile   |record of old file :  old file (index).record .
1529
1530  checkforinterrupt |check for interrupt :
1531                    |  INT VAR size, used ;
1532                    |  storage (size, used) ;
1533                    |  IF used > size
1534                    |  THEN errorstop ("Speicherengpass")
1535                    |  FI ;
1536                    |  IF is error
1537                    |  THEN forget (scratch space) ; LEAVE reorganize
1538                    |  FI .
1539
1540  replacefileospacebyscra |replace file space by scratch space :
1541                    |  headline (output file, file name);
1542                    |  forget (file name, quiet) ;
1543                    |  type (scratch space, file type);
1544                    |  copy (scratch space, file name);
1545                    |  forget (scratch space) .
1546
1547                    |END PROC reorganize;
1548
1549

```

```

1550 setrange .....|PROC set range (FILE VAR f, INT CONST start line, start col,
1551                |          FRANGE VAR old range) :
1552
1553                |   check mode (f, mod);
1554                |   IF valid restriction parameters
1555                |       THEN prepare last line ;
1556                |           prepare first line ;
1557                |           save old range ;
1558                |           set new range
1559                |       ELSE errorstop ("FRANGE ungueltig")
1560                |   FI .
1561
1562 validrestrictionparame|valid restriction parameters :
1563                |   start line > 0 AND start col > 0 AND start before or at actual
+                |   point .
1564
1565 startbeforeoratactualp|start before or at actual point :
1566                |   start line < line no (f) OR
1567                |   start line = line no (f) AND start col <= col (f) .
1568
1569 preparelastline       |prepare last line :
1570                |   INT VAR last line ;
1571                |   IF col (f) > 1
1572                |       THEN split line (f, col(f), FALSE)
1573                |   FI .
1574
1575 preparefirstline      |prepare first line :
1576                |   IF start col > 1
1577                |       THEN split start line ;
1578                |   FI .
1579
1580 splitstartline        |split start line :
1581                |   INT VAR old line no := line no (f) ;
1582                |   to line (f, start line) ;
1583                |   split line (f, start col, FALSE) ;
1584                |   to line (f, old line no + 1) .
1585
1586 saveoldrange          |save old range :
1587                |   old range.pre := f.prefix lines ;
1588                |   old range.post:= f.postfix lines .
1589
1590 setnewrange           |set new range :
1591                |   get pre lines ;
1592                |   get post lines ;
1593                |   disable stop ;
1594                |   f.prefix lines INCR pre lines ;
1595                |   f.postfix lines INCR post lines ;
1596                |   f.used.lines DECR (post lines + pre lines) ;
1597                |   f.used.line no DECR pre lines .
1598
1599 getprelines           |get pre lines :
1600                |   INT VAR pre lines ;
1601                |   IF start col = 1
1602                |       THEN old range.pre was split := FALSE ;

```

```

1603             pre lines := start line - 1
1604             ELSE old range.pre was split := TRUE ;
1605             pre lines := start line
1606         FI .
1607
1608     getpostlines   get post lines :
1609                   INT VAR post lines ;
1610                   IF col (f) = 1
1611                       THEN old range.post was split := FALSE ;
1612                            post lines := lines (f) - line no (f) + 1
1613                   ELSE old range.post was split := TRUE ;
1614                            post lines := lines (f) - line no (f)
1615                   FI .
1616
1617     END PROC set range;
1618
1619

```

```

1620 setrange ..... PROC set range (FILE VAR f, FRANGE VAR new range) :
1621
1622             check mode (f, mod);
1623             INT CONST pre add := prefix - new range.pre,
1624                    post add := postfix - new range.post;
1625             IF pre add < 0 OR post add < 0
1626                 THEN errorstop ("FRANGE ungueltig")
1627             ELSE set new range;
1628                    undo splitting if necessary ;
1629                    make range var invalid
1630             FI .
1631

```

```

1632 setnewrange     set new range :
1633                 disable stop;
1634                 prefix DECR pre add;
1635                 postfix DECR post add;
1636                 used.line no INCR pre add;
1637                 used.lines INCR (pre add + post add) .
1638

```

```

1639 undosplittingifnecessa  undo splitting if necessary :
1640                         IF new range.pre was split
1641                             THEN concatenate first line
1642                         FI ;
1643                         IF new range.post was split
1644                             THEN concatenate last line
1645                         FI .
1646

```

```

1647 concatenatefirstline    concatenate first line :
1648                         INT VAR old line := line no (f) ;
1649                         to line (f, pre add) ;
1650                         concatenate line (f, FALSE) ;
1651                         to line (f, old line - 1) .
1652

```

```

1653 concatenatelastline     concatenate last line :
1654                         old line := line no (f) ;
1655                         to line (f, lines (f) - post add) ;
1656                         concatenate line (f, FALSE) ;
1657                         to line (f, old line) .

```

```

1658
1659     makerangevarinvalid |make range var invalid :
1660                          |new range.pre := maxint .
1661
1662     used                  |used :   f.used .
1663     prefix                |prefix : f.prefix lines .
1664     postfix               |postfix : f.postfix lines .
1665
1666                          |END PROC set range;
1667
1668     resetrange ..... |PROC reset range (FILE VAR f) :
1669                          |
1670                          |FRANGE VAR complete ;
1671                          |complete.pre := @ ;
1672                          |complete.post:= @ ;
1673                          |complete.pre was split := FALSE ;
1674                          |complete.post was split:= FALSE ;
1675                          |set range (f, complete)
1676
1677                          |ENDPROC reset range ;
1678
1679     remove ..... |PROC remove (FILE VAR f, INT CONST size) :
1680                          |
1681                          |check mode (f, mod);
1682                          |transfer subsequence (f.used, f.scratch, f.atoms, size) .
1683
1684                          |END PROC remove;
1685
1686
1687     clearremoved ..... |PROC clear removed (FILE VAR f) :
1688                          |
1689                          |check mode (f, mod);
1690                          |transfer subsequence (f.scratch, f.free, f.atoms, f.scratch.lines)
1691                          |
1692                          |
1693                          |END PROC clear removed;
1694
1695     reinsert ..... |PROC reinsert (FILE VAR f) :
1696                          |
1697                          |check mode (f, mod);
1698                          |transfer subsequence (f.scratch, f.used, f.atoms, f.scratch.lines)
1699                          |
1700                          |
1701                          |END PROC reinsert;
1702

```

```

1703 copyattributes .....|PROC copy attributes (FILE CONST source file, FILE VAR dest file) :
1704                       |
1705                       |   dest.limit                := source.limit ;
1706                       |   dest.atoms (free root).line := source.atoms (free root).line ;
1707                       |   dest.atoms (scratch root).line := source.atoms (scratch root).line
1708                       |   ;
1709                       |   dest.edit info                := source.edit info .
1710
1710     dest                |dest  : CONCR (CONCR (dest file)) .
1711
1711     source              |source : CONCR (CONCR (source file)) .
1712
1713                       |ENDPROC copy attributes ;
1714
1715                       |
1716
1716 maxlinelength .....|INT PROC max line length (FILE CONST f) :
1717                       |
1718                       |   f.limit .
1719
1720                       |END PROC max line length;
1721
1722                       |
1723
1723 maxlinelength .....|PROC max line length (FILE VAR f, INT CONST new limit) :
1724                       |
1725                       |   IF new limit > 0 AND new limit <= max limit
1726                       |     THEN f.limit := new limit
1727                       |   FI .
1728
1729                       |END PROC max line length;
1730
1731                       |
1732
1732 headline .....|TEXT PROC headline (FILE CONST f) :
1733                       |
1734                       |   f.atoms (free root).line .
1735
1736                       |END PROC headline;
1737
1738                       |
1739
1739 headline .....|PROC headline (FILE VAR f, TEXT CONST head) :
1740                       |
1741                       |   f.atoms (free root).line := head .
1742
1743                       |END PROC headline;
1744
1745                       |
1746
1746 gettabs .....|PROC get tabs (FILE CONST f, TEXT VAR tabs) :
1747                       |
1748                       |   tabs := f.atoms (scratch root).line .
1749
1750                       |END PROC get tabs;
1751
1751                       |

```

1752

```

1753 puttabs .....|PROC put tabs (FILE VAR f, TEXT CONST tabs) :
1754                |
1755                |   f.atoms (scratch root).line := tabs .
1756                |
1757                |END PROC put tabs;
1758                |
1759                |
    
```

```

1760 editinfo .....|INT PROC edit info (FILE CONST f) :
1761                |
1762                |   f.edit info .
1763                |
1764                |END PROC edit info;
1765                |
1766                |
    
```

```

1767 editinfo .....|PROC edit info (FILE VAR f, INT CONST info) :
1768                |
1769                |   f.edit info := info .
1770                |
1771                |END PROC edit info;
1772                |
1773                |
    
```

```

1774 lines .....|INT PROC lines (FILE CONST f) :
1775                |
1776                |   f.used.lines .
1777                |
1778                |END PROC lines;
1779                |
1780                |
    
```

```

1781 removedlines .....|INT PROC removed lines (FILE CONST f) :
1782                |
1783                |   f.scratch.lines .
1784                |
1785                |END PROC removed lines;
1786                |
1787                |
    
```

```

1788 segments .....|INT PROC segments (FILE CONST f) :
1789                |
1790                |   segs(f.used,f.atoms) + segs(f.scratch,f.atoms) +
1791                |       +
1792                |       segs(f.free,f.atoms) - 2 .
1793                |ENDPROC segments ;
1794                |
    
```

```

1795 col .....|INT PROC col (FILE CONST f) :
1796                |
1797                |   f.col
1798                |
    
```

```

1799          |ENDPROC col ;
1800          |

1801 col .....|PROC col (FILE VAR f, INT CONST new column) :
1802          |
1803          |   IF new column > 0
1804          |     THEN f.col := new column
1805          |   FI
1806          |
1807          |ENDPROC col ;
1808          |

1809 word .....|TEXT PROC word (FILE CONST f) :
1810          |
1811          |   word (f, " ")
1812          |
1813          |ENDPROC word ;
1814          |

1815 word .....|TEXT PROC word (FILE CONST f, TEXT CONST delimiter) :
1816          |
1817          |   INT VAR del pos := pos (f, delimiter, col (f)) ;
1818          |   IF del pos = 0
1819          |     THEN del pos := len (f) + 1
1820          |   FI ;
1821          |   subtext (f, col (f), del pos - 1)
1822          |
1823          |ENDPROC word ;
1824          |

1825 word .....|TEXT PROC word (FILE CONST f, INT CONST max length) :
1826          |
1827          |   subtext (f, col (f), col (f) + max length - 1)
1828          |
1829          |ENDPROC word ;
1830          |

1831 at .....|BOOL PROC at (FILE CONST f, TEXT CONST word) :
1832          |
1833          |   pat := any (column-1) ;
1834          |   pat CAT word ;
1835          |   pat CAT any ;
1836          |   record LIKE pat .
1837          |

1838 column    |column : f.col .
1839 record    |record : f.atoms (f.used.index).line .
1840          |
1841          |ENDPROC at ;
1842          |
1843          |

1844 exec .....|PROC exec (PROC (TEXT VAR, TEXT CONST) proc, FILE VAR f, TEXT CONST
+          |   t) :
1845          |

```

```

1846          | proc (record, t) .
1847          |
1848 record    | record : f.atoms (f.used.index).line .
1849          |
1850          | END PROC exec;
1851          |
1852          |
1853 exec .....| PROC exec (PROC (TEXT VAR, INT CONST) proc, FILE VAR f, INT CONST i)
+          | :
1854          |
1855          |   proc (record, i) .
1856          |
1857 record    | record : f.atoms (f.used.index).line .
1858          |
1859          | END PROC exec;
1860          |
1861 pos .....| INT PROC pos (FILE CONST f, TEXT CONST pattern, INT CONST i) :
1862          |
1863          |   pos (record, pattern, i) .
1864          |
1865 record    | record : f.atoms (f.used.index).line .
1866          |
1867          | END PROC pos ;
1868          |
1869 down .....| PROC down (FILE VAR f, TEXT CONST pattern) :
1870          |
1871          |   down (f, pattern, file size)
1872          |
1873          | ENDPROC down ;
1874          |
1875 down .....| PROC down (FILE VAR f, TEXT CONST pattern, INT CONST max line) :
1876          |
1877          |   check mode (f,mod) ;
1878          |   INT VAR pattern pos := f.col + 1 ;
1879          |   search down (f.used, f.atoms, pattern, max line, pattern pos) ;
1880          |   f.col := pattern pos
1881          |
1882          | ENDPROC down ;
1883          |
1884 downety .....| PROC downety (FILE VAR f, TEXT CONST pattern) :
1885          |
1886          |   downety (f, pattern, file size)
1887          |
1888          | ENDPROC downety ;
1889          |

```

```

1890 downety .....|PROC downety (FILE VAR f, TEXT CONST pattern, INT CONST max line)
1891                |
1892                |   check mode (f,mod) ;
1893                |   INT VAR pattern pos := f.col ;
1894                |   search down (f.used, f.atoms, pattern, max line, pattern pos) ;
1895                |   f.col := pattern pos
1896                |
1897                |ENDPROC downety ;
1898                |

1899 up .....|PROC up (FILE VAR f, TEXT CONST pattern) :
1900        |
1901        |   up (f, pattern, file size)
1902        |
1903        |ENDPROC up ;
1904        |

1905 up .....|PROC up (FILE VAR f, TEXT CONST pattern, INT CONST max line) :
1906        |
1907        |   check mode (f,mod) ;
1908        |   INT VAR pattern pos := f.col - 1 ;
1909        |   search up (f.used, f.atoms, pattern, max line, pattern pos) ;
1910        |   f.col := pattern pos
1911        |
1912        |ENDPROC up ;
1913        |

1914 uppety .....|PROC uppety (FILE VAR f, TEXT CONST pattern) :
1915        |
1916        |   uppety (f, pattern, file size)
1917        |
1918        |ENDPROC uppety ;
1919        |

1920 uppety .....|PROC uppety (FILE VAR f, TEXT CONST pattern, INT CONST max line)
1921                |
1922                |   check mode (f,mod) ;
1923                |   INT VAR pattern pos := f.col ;
1924                |   search up (f.used, f.atoms, pattern, max line, pattern pos) ;
1925                |   f.col := pattern pos
1926                |
1927                |ENDPROC uppety ;
1928                |
1929                |

1930 len .....|INT PROC len (FILE CONST f) :
1931        |
1932        |   length (record) .
1933        |

1934   record   |record : f.atoms (f.used.index).line .
1935        |
1936        |ENDPROC len ;
1937        |

```

```

1938 subtext .....|TEXT PROC subtext (FILE CONST f, INT CONST from, to) :
1939                |
1940                |   subtext (record, from, to) .
1941                |
1942   record         |record : f.atoms (f.used.index).line .
1943                |
1944                |ENDPROC subtext ;
1945                |

```

```

1946 change .....|PROC change (FILE VAR f, INT CONST from, to, TEXT CONST new) :
1947                |
1948                |   check mode (f, mod) ;
1949                |   change (record, from, to, new) .
1950                |
1951   record         |record : f.atoms (f.used.index).line .
1952                |
1953                |ENDPROC change ;
1954                |
1955                |

```

```

1956 mark .....|BOOL PROC mark (FILE CONST f) :
1957                |
1958                |   f.mark line > 0
1959                |
1960                |ENDPROC mark ;
1961                |

```

```

1962 mark .....|PROC mark (FILE VAR f, INT CONST line no, col) :
1963                |
1964                |   IF line no > 0
1965                |     THEN f.mark line := line no + f.prefix lines ;
1966                |           f.mark col := col
1967                |     ELSE f.mark line := 0 ;
1968                |           f.mark col := 0
1969                |   FI
1970                |
1971                |ENDPROC mark ;
1972                |

```

```

1973 marklineno .....|INT PROC mark line no (FILE CONST f) :
1974                |
1975                |   IF f.mark line = 0
1976                |     THEN 0
1977                |     ELSE max (1, f.mark line - f.prefix lines)
1978                |   FI
1979                |
1980                |ENDPROC mark line no ;
1981                |

```

```

1982 markcol .....|INT PROC mark col (FILE CONST f) :
1983                |
1984                |   IF f.mark line = 0
1985                |     THEN 0
1986                |     ELIF f.mark line <= f.prefix lines
1987                |     THEN 1

```

```

1988 | ELSE f.mark col
1989 | FI
1990 |
1991 |ENDPROC mark col ;
1992 |

1993 setmarkedrange .....|PROC set marked range (FILE VAR f, FRANGE VAR old range) :
1994 |
1995 | IF mark (f)
1996 | THEN set range (f, mark line no (f), mark col (f), old range)
1997 | ELSE old range := previous range of file
1998 | FI .
1999 |

2000 previousrangeoffile |previous range of file :
2001 | FRANGE : (f.prefix lines, f.postfix lines, FALSE, FALSE) .
2002 |
2003 |ENDPROC set marked range ;
2004 |
2005 |
2006 | (*****
2007 |
2008 |
2009 | P.Heyderhoff *) (* Autor:
2010 | *) (* Stand: 11.10.83
2011 |
2012 | BOUND LIST VAR datei;
2013 | INT VAR sortierstelle, sortanker;
2014 | BOOL VAR ascii sort;
2015 | TEXT VAR median, tausch , links, rechts;

2016 sort .....|PROC sort (TEXT CONST dateiname) :
2017 | sort (dateiname, 1)
2018 |END PROC sort;
2019 |

2020 sort .....|PROC sort (TEXT CONST dateiname, INT CONST sortieranfang) :
2021 | ascii sort := TRUE ;
2022 | sortierstelle := sortieranfang; sortiere (dateiname)
2023 |END PROC sort;
2024 |

2025 lexsort .....|PROC lex sort (TEXT CONST dateiname) :
2026 | lex sort (dateiname, 1)
2027 |ENDPROC lex sort ;
2028 |

2029 lexsort .....|PROC lex sort (TEXT CONST dateiname, INT CONST sortieranfang) :
2030 | ascii sort := FALSE ;
2031 | sortierstelle := sortieranfang; sortiere (dateiname)
2032 |ENDPROC lex sort ;
2033 |

```

```

2034 sortiere .....|PROC sortiere (TEXT CONST dateiname) :
2035
2036 |         reorganize file if necessary ;
2037 |         sort file .
2038
2039 reorganizefileifnecess |reorganize file if necessary :
2040 | FILE VAR f := sequential file (modify, dateiname) ;
2041 | IF segments (f) > 1
2042 |     THEN reorganize (dateiname)
2043 |     FI .
2044
2045 sortfile |sort file :
2046 | f := sequential file (modify, dateiname) ;
2047 | INT CONST sortende := lines (f) + 3 ;
2048 | sortanker := 1 + 3 ;
2049 | datei := old (dateiname) ;
2050 | quicksort(sortanker, sortende) .
2051
2052 |END PROC sortiere;
2053
2054 quicksort .....|PROC quicksort ( INT CONST anfang, ende ) :
2055 |     IF anfang < ende
2056 |     THEN INT VAR p,q;
2057 |         spalte (anfang, ende, p, q);
2058 |         quicksort (anfang, q);
2059 |         quicksort (p, ende)  FI
2060 |END PROC quicksort;
2061
2062 spalte .....|PROC spalte (INT CONST anfang, ende, INT VAR p, q):
2063 | fange an der seite an und waehle den median;
2064 | ruecke p und q so dicht wie moeglich zusammen;
2065 | hole ggf median in die mitte .
2066
2067 fangeanderseiteanundwa |fange an der seite an und waehle den median :
2068 | p := anfang; q := ende ;
2069 | INT CONST m :: (p + q) DIV 2 ;
2070 | median := subtext(datei m, sortierstelle) .
2071
2072 rueckepundqsodichtwiem |ruecke p und q so dicht wie moeglich zusammen :
2073 | REP schiebe p und q so weit wie moeglich auf bzw ab;
2074 |     IF p < q THEN vertausche die beiden FI
2075 |     UNTIL p > q END REP .
2076
2077 vertauschediebeiden |vertausche die beiden :
2078 | tausch := datei p; datei p := datei q; datei q := tausch;
2079 | p INCR 1; q DECR 1 .
2080
2081 schiebepundqsoweitwiem |schiebe p und q so weit wie moeglich auf bzw ab :
2082 | WHILE p kann groesser werden REP p INCR 1 END REP;
2083 | WHILE q kann kleiner werden REP q DECR 1 END REP .
2084

```

```

2085   pkanngroesserwerden      | p kann groesser werden :
2086                               | IF p <= ende
2087                               |   THEN links := subtext (datei p, sortierstelle) ;
2088                               |     IF ascii sort
2089                               |       THEN median >= links
2090                               |         ELSE median LEXGREATEREQUAL links
2091                               |     FI
2092                               |   ELSE FALSE
2093                               | FI .
2094
2095   qkannkleinerwerden        | q kann kleiner werden :
2096                               | IF q >= anfang
2097                               |   THEN rechts := subtext(datei q, sortierstelle) ;
2098                               |     IF ascii sort
2099                               |       THEN rechts >= median
2100                               |         ELSE rechts LEXGREATEREQUAL median
2101                               |     FI
2102                               |   ELSE FALSE
2103                               | FI .
2104
2105   holeggfmedianindiemitt    | hole ggf median in die mitte :
2106                               | IF m < q THEN vertausche m und q
2107                               | ELIF m > p THEN vertausche m und p FI .
2108
2109   vertauschemundq           | vertausche m und q :
2110                               |   tausch := datei m; datei m := datei q; datei q := tausch; q
2111                               |     +
2112                               |     DECR 1 .
2113
2114   vertauschemundp           | vertausche m und p :
2115                               |   tausch := datei m; datei m := datei p; datei p := tausch; p
2116                               |     +
2117                               |     INCR 1 .
2118
2119   dateim                    | datei m : datei.atoms (m).line .
2120   dateip                    | datei p : datei.atoms (p).line .
2121   dateiq                    | datei q : datei.atoms (q).line .
2122
2123                               | END PROC spalte;
2124                               | END PACKET file handling;

```

```

1
2 elandointerface *****|PACKET elan do interface DEFINES (*Autor: J.Lied
+ *)
3 (*Stand: 08.11.85
+ *)
4 do ,
5 no do again :
6
7
8 |LET no ins = FALSE ,
9 | no lst = FALSE ,
10 | no check = FALSE ,
11 | no sermon = FALSE ,
12 | compile line mode = 2 ,
13 | do again mode = 4 ,
14 | max command length = 2000 ;
15
16
17 |INT VAR do again mod nr := 0 ;
18 |TEXT VAR previous command := "" ;
19
20 |DATASPACE VAR ds ;
21
22
23 do .....|PROC do (TEXT CONST command) :
24
25 | enable stop ;
26 | IF LENGTH command > max command length
27 | THEN errorstop ("Kommando zu lang")
28 | ELIF do again mod nr <> 0 AND command = previous command
29 | THEN do again
30 | ELSE previous command := command ;
31 | compile and execute
32 | FI .
33
34 doagain |do again :
35 | elan (do again mode, ds, "", do again mod nr,
36 | no ins, no lst, no check, no sermon) .
37
38 compileandexecute |compile and execute :
39 | elan (compile line mode, ds, command, do again mod nr,
40 | no ins, no lst, no check, no sermon) .
41
42 |ENDPROC do ;
43
44 nodoagain .....|PROC no do again :
45
46 | do again mod nr := 0
47
48 |ENDPROC no do again ;
49
50 elan .....|PROC elan (INT CONST mode, DATASPACE CONST source, TEXT CONST line,
51 | INT VAR start module number,
52 | BOOL CONST ins, lst, rt check, ser) :
```

Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** elan do interface

```
53 | EXTERNAL 256
54 |ENDPROC elan ;
55 |
56 |ENDPACKET elan do interface ;
```

```

1      | (* ----- VERSION 4      14.05.86 ----- *)
2  scanner *****|PACKET scanner DEFINES      (* Autor: J.Liedtke *)
3      |
4      |         scan ,
5      |         continue scan ,
6      |         next symbol :
7      |
8      |
9      |   LET tag      = 1 ,
10     |   bold       = 2 ,
11     |   number     = 3 ,
12     |   text       = 4 ,
13     |   operator   = 5 ,
14     |   delimiter  = 6 ,
15     |   end of file = 7 ,
16     |   within comment = 8 ,
17     |   within text  = 9 ;
18     |
19     |   LET digit 0   = 48 ,
20     |   digit 9     = 57 ,
21     |   upper case a = 65 ,
22     |   upper case z = 90 ,
23     |   lower case a = 97 ,
24     |   lower case z = 122;
25     |
26     |
27     |   TEXT VAR line := "" ,
28     |   char := "" ,
29     |   chars:= "" ;
30     |
31     |   INT VAR position := 0 ,
32     |   comment depth ;
33     |   BOOL VAR continue text ;
34     |
35     |
36     |
37     |   PROC scan (TEXT CONST scan text) :
38     |   comment depth := 0 ;
39     |   continue text := FALSE ;
40     |   continue scan (scan text)
41     |
42     |   ENDPROC scan ;
43     |
44     |
45     |   PROC continue scan (TEXT CONST scan text) :
46     |   line := scan text ;
47     |   position := 0 ;
48     |   nextchar
49     |
50     |   ENDPROC continue scan ;
51     |
52     |
53     |   PROC next symbol (TEXT VAR symbol) :
54     |   INT VAR type ;
55     |   next symbol (symbol, type)
56     |

```

```

57                                     |ENDPROC next symbol ;
58                                     |
59 nextsymbol .....|PROC next symbol (TEXT VAR symbol, INT VAR type) :
60                                     |
61                                     | skip blanks ;
62                                     | IF is begin comment          THEN process comment
63                                     | ELIF comment depth > 0      THEN comment depth DECR 1 ;
64                                     |                             process comment
65                                     | ELIF is quote OR continue text THEN process text
66                                     | ELIF is lower case letter    THEN process tag
67                                     | ELIF is upper case letter    THEN process bold
68                                     | ELIF is digit                THEN process number
69                                     | ELIF is delimiter           THEN process delimiter
70                                     | ELIF is niltext             THEN eof
71                                     | ELSE process operator
72                                     | FI .
73
74
75 processcomment |process comment :
76               | read comment ;
77               | IF comment depth = 0
78               |     THEN next symbol (symbol, type)
79               |     ELSE type := within comment ;
80               |         symbol := ""
81               | FI .
82
83 processtag    |process tag :
84               | type := tag ;
85               | assemble chars (lower case a, lower case z) ;
86               | symbol := chars ;
87               | REP
88               |     skip blanks ;
89               |     IF is lower case letter
90               |         THEN assemble chars (lower case a, lower case z)
91               |     ELIF is digit
92               |         THEN assemble chars (digit 0, digit 9)
93               |     ELSE LEAVE process tag
94               |     FI ;
95               |     symbol CAT chars
96               | PER ;
97               | nextchar .
98
99 processbold   |process bold :
100              | type := bold ;
101              | assemble chars (upper case a, upper case z) ;
102              | symbol := chars .
103
104 processnumber |process number :
105              | type := number ;
106              | assemble chars (digit 0, digit 9) ;
107              | symbol := chars ;
108              | IF char = "." AND ahead char is digit
109              |     THEN process fraction ;
110              |         IF char = "e"
111              |             THEN process exponent
112              |     FI

```

```

113 | FI .
114 |
115 | aheadcharisdigit | ahead char is digit :
116 |                 | digit 0 <= code (ahead char) AND code (ahead char) <= digit 9 .
117 |
118 | processfraction | process fraction :
119 |                 | symbol CAT char ;
120 |                 | nextchar ;
121 |                 | assemble chars (digit 0, digit 9) ;
122 |                 | symbol CAT chars .
123 |
124 | processexponent | process exponent :
125 |                 | symbol CAT char ;
126 |                 | nextchar ;
127 |                 | IF char = "+" OR char = "-"
128 |                 | THEN symbol CAT char ;
129 |                 | nextchar
130 |                 | FI ;
131 |                 | assemble chars (digit 0, digit 9) ;
132 |                 | symbol CAT chars .
133 |
134 | processtext     | process text :
135 |                 | type := text ;
136 |                 | symbol := "" ;
137 |                 | IF continue text
138 |                 | THEN continue text := FALSE
139 |                 | ELSE next char
140 |                 | FI ;
141 |                 | WHILE not end of text REP
142 |                 | assemble chars (35, 254) ;
143 |                 | symbol CAT chars ;
144 |                 | IF NOT is quote
145 |                 | THEN symbol CAT char ;
146 |                 | nextchar
147 |                 | FI
148 |                 | ENDREP .
149 |
150 | notendoftext   | not end of text :
151 |                 | IF is niltext
152 |                 | THEN continue text := TRUE ; type := within text ; FALSE
153 |                 | ELIF is quote
154 |                 | THEN end of text or exception
155 |                 | ELSE TRUE
156 |                 | FI .
157 |
158 | endoftextorexception | end of text or exception :
159 |                 | next char ;
160 |                 | IF is quote
161 |                 | THEN get quote ; TRUE
162 |                 | ELIF is digit
163 |                 | THEN get special char ; TRUE
164 |                 | ELSE FALSE
165 |                 | FI .
166 |

```

```

167 getquote |get quote :
168 | symbol CAT char ;
169 | nextchar .
170
171 getspecialchar |get special char :
172 | assemble chars (digit 0, digit 9) ;
173 | symbol CAT code (int (chars) ) ;
174 | nextchar .
175
176 processdelimiter |process delimiter :
177 | type := delimiter ;
178 | symbol := char ;
179 | nextchar .
180
181 processoperator |process operator :
182 | type := operator ;
183 | symbol := char ;
184 | nextchar ;
185 | IF symbol = ":"
186 | THEN IF char = "=" OR char = ":"
187 | THEN symbol := "!=" ;
188 | nextchar
189 | ELSE type := delimiter
190 | FI
191 | ELIF is relational double char
192 | THEN symbol CAT char ;
193 | nextchar
194 | ELIF symbol = "*" AND char = "*"
195 | THEN symbol := "***" ;
196 | next char
197 | FI .
198
199 eof |eof :
200 | type := end of file ;
201 | symbol := "" .
202
203 islowercaseletter |is lower case letter :
204 | lower case a <= code (char) AND code (char) <= lower case z .
205
206 isuppercaseletter |is upper case letter :
207 | upper case a <= code (char) AND code (char) <= upper case z .
208
209 isdigit |is digit :
210 | digit 0 <= code (char) AND code (char) <= digit 9 .
211
212 isdelimiter |is delimiter : pos ( "()[],;" , char ) > 0 .
213
214 isrelationaldoublechar |is relational double char :
215 | TEXT VAR double := symbol + char ;
216 | double = "<>" OR double = "<=" OR double = ">=" .
217

```

```

218 isquote          | is quote : char = "" .
219                |
220 isniltext        | is niltext : char = "" .
221                |
222 isbegincomment    | is begin comment : char = "{" OR char = "(" AND ahead char = "*" .
223                |
224                | ENDPROC next symbol ;
225                |

226 nextchar ..... | PROC next char :
227                |
228                |   position INCR 1 ;
229                |   char := line SUB position
230                |
231                | ENDPROC next char ;
232                |

233 skipblanks ..... | PROC skip blanks :
234                |
235                |   position := pos (line, ""33"", ""254"", position) ;
236                |   IF position = 0
237                |     THEN position := LENGTH line + 1
238                |   FI ;
239                |   char := line SUB position .
240                |
241                | ENDPROC skip blanks ;
242                |

243 aheadchar ..... | TEXT PROC ahead char :
244                |
245                |   line SUB position+1
246                |
247                | ENDPROC ahead char ;
248                |

249 assemblechars ..... | PROC assemble chars (INT CONST low, high) :
250                |
251                |   INT CONST begin := position ;
252                |   position behind valid text ;
253                |   chars := subtext (line, begin, position-1) ;
254                |   char := line SUB position .
255                |

256 positionbehindvalidtex | position behind valid text :
257                |   position := pos (line, ""32"", code (low-1), begin) ;
258                |   IF position = 0
259                |     THEN position := LENGTH line + 1
260                |   FI ;
261                |   INT CONST higher pos := pos (line, code (high+1), ""254"", begin) ;
262                |   IF higher pos <> 0 AND higher pos < position
263                |     THEN position := higher pos
264                |   FI .
265                |
266                | ENDPROC assemble chars ;
267                |
268                |

```

```

269 readcomment .....|PROC read comment :
270                    |
271                    |   TEXT VAR last char ;
272                    |   comment depth INCR 1 ;
273                    |   REP
274                    |     last char := char ;
275                    |     nextchar ;
276                    |     IF is begin comment
277                    |       THEN read comment
278                    |     FI ;
279                    |     IF char = ""
280                    |       THEN LEAVE read comment
281                    |     FI
282                    |   UNTIL is end comment PER ;
283                    |   comment depth DECR 1 ;
284                    |   next char ;
285                    |   skip blanks .
286                    |
287 isendcomment        |is end comment :
288                    | char = ")" OR char = "" AND last char = "*" .
289                    |
290 isbegincomment      |is begin comment :
291                    | char = "{" OR char = "(" AND ahead char = "*" .
292                    |
293                    |ENDPROC read comment ;
294                    |
295                    |
296 scan .....|PROC scan (FILE VAR f) :
297            |
298            |   getline (f, line) ;
299            |   scan (line)
300            |
301            |ENDPROC scan ;
302            |
303 nextsymbol .....|PROC next symbol (FILE VAR f, TEXT VAR symbol) :
304            |
305            |   INT VAR type ;
306            |   next symbol (f, symbol, type)
307            |
308            |ENDPROC next symbol ;
309            |
310            |TEXT VAR scanned ;
311            |
312 nextsymbol .....|PROC next symbol (FILE VAR f, TEXT VAR symbol, INT VAR type) :
313            |
314            |   next symbol (symbol, type) ;
315            |   WHILE type >= 7 AND NOT eof (f) REP
316            |     getline (f, line) ;
317            |     continue scan (line) ;
318            |     next symbol (scanned, type) ;
319            |     symbol CAT scanned
320            |   PER .
321            |
322            |ENDPROC next symbol ;

```

Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** scanner

323
324

|
|ENDPACKET scanner ;

```

1      |
2  screendescription *****|PACKET screen description DEFINES
3      |
4      |           xsize, ysize, marksize, mark refresh line mode :
5      |
6      |
7      |INT VAR xs := 80, ys := 24, ms := 1;
8      |
9      |
9  xsize .....|INT PROC xsize: xs END PROC xsize;
10     |
11     |
11  ysize .....|INT PROC ysize: ys END PROC ysize;
12     |
13     |
13  marksize .....|INT PROC marksize: ms END PROC marksize;
14     |
15     |
15  xsize .....|PROC xsize (INT CONST i): xs := i END PROC xsize;
16     |
17     |
17  ysize .....|PROC ysize (INT CONST i): ys := i END PROC ysize;
18     |
19     |
19  marksize .....|PROC marksize (INT CONST i): ms := i END PROC marksize;
20     |
21     |
22     |          BOOL VAR line mode := FALSE;
23     |
24     |
24  markrefreshlinemode .....|BOOL PROC mark refresh line mode:
25     |          line mode
26     |          END PROC mark refresh line mode;
27     |
28     |
28  markrefreshlinemode .....|PROC mark refresh line mode (BOOL CONST b):
29     |          line mode := b
30     |          END PROC mark refresh line mode;
31     |
32     |          END PACKET screen description ;

```

```

1
2 tastenverwaltung *****|PACKET tastenverwaltung DEFINES
+                            |#009 *)
3                            |(*****
4
5                            |lernsequenz auf taste legen,
6                            |lernsequenz auf taste,
7                            |kommando auf taste legen,
8                            |kommando auf taste,
9                            |taste enthaelt kommando,
10                           |std tastenbelegung :
11
12
13
14                           |LET kommandoidentifikation = ""0" ,
15                           |esc = ""27" ,
16                           |niltext = "" ,
17                           |hop right left up down cr tab rubin rubout mark esc
18                           |= ""1""2""8""3""10""13""9""11""12""16""27"" ;
19
20
21                           |ROW 256 TEXT VAR belegung;
22                           |INT VAR i; FOR i FROM 1 UPTO 256 REP belegung (i) := "" PER;
23
24                           |std tastenbelegung;
25
26
27 lernsequenzauftasteleg ...|PROC lernsequenz auf taste legen (TEXT CONST taste, lernsequenz) :
28                             |
29                             |belege (belegung (code (taste) + 1), taste, lernsequenz)
30
31                             |ENDPROC lernsequenz auf taste legen ;
32
33
34 belege .....|PROC belege (TEXT VAR tastenpuffer, TEXT CONST taste, lernsequenz) :
35               |tastenpuffer := lernsequenz ;
36               |verhindere rekursives lernen .
37
38   verhindererekursivesle |verhindere rekursives lernen :
39                             |loesche alle folgen esc taste aber nicht esc esc taste ;
40                             |IF taste ist freies sonderzeichen
41                             |THEN change all (tastenpuffer, taste, niltext)
42                             |FI .
43
44   loescheallefolgenescta |loesche alle folgen esc taste aber nicht esc esc taste :
45                             |INT VAR i := pos (tastenpuffer, esc + taste) ;
46                             |WHILE i > 0 REP
47                             |IF ist esc esc taste
48                             |THEN i INCR 1
49                             |ELSE change (tastenpuffer, i, i+1, niltext)
50                             |FI ;
51                             |i := pos (tastenpuffer, esc + taste, i)
52                             |PER .

```

```

53  istescstaste          | ist esc esc taste :
54                        | ( tastenpuffer SUB i-1) = esc AND ( tastenpuffer SUB i-2) <> esc .
55                        |
56  tasteistfreiessonderze | taste ist freies sonderzeichen :
57                        | taste < ""32"" AND
58                        | pos (hop right left up down cr tab rubin rubout mark esc, taste)
59                        | @ .
60                        |
61                        | END PROC belege ;
62                        |
63  lernsequenzauf taste ..... | TEXT PROC lernsequenz auf taste (TEXT CONST taste) :
64                        | IF taste enthaelt kommando (taste)
65                        | THEN ""
66                        | ELSE belegung (code (taste) + 1)
67                        | FI
68                        | END PROC lernsequenz auf taste;
69                        |
70                        |
71  kommandoauftastelegen .... | PROC kommando auf taste legen (TEXT CONST taste, kommando) :
72                        |
73                        |   belegung (code (taste) + 1) := kommandoidentifikation;
74                        |   belegung (code (taste) + 1) CAT kommando
75                        |
76                        | END PROC kommando auf taste legen;
77                        |
78                        |
79  kommandoauftaste ..... | TEXT PROC kommando auf taste (TEXT CONST taste) :
80                        | IF taste enthaelt kommando (taste)
81                        | THEN subtext (belegung (code (taste) + 1), 2)
82                        | ELSE ""
83                        | FI
84                        | END PROC kommando auf taste;
85                        |
86                        |
87  tasteenthaeltkommando .... | BOOL PROC taste enthaelt kommando (TEXT CONST taste) :
88                        | (belegung (code (taste) + 1) SUB 1) = kommandoidentifikation
89                        | END PROC taste enthaelt kommando;
90                        |
91                        |
92  stdtastenbelegung ..... | PROC std tastenbelegung:
93                        | lernsequenz auf taste legen ("", ""91"");
94                        | lernsequenz auf taste legen ("", ""93"");
95                        | lernsequenz auf taste legen (< "", ""123"");
96                        | lernsequenz auf taste legen (> "", ""125"");
97                        | lernsequenz auf taste legen ("A", ""214"");
98                        | lernsequenz auf taste legen ("O", ""215"");
99                        | lernsequenz auf taste legen ("U", ""216"");
100                       | lernsequenz auf taste legen ("a", ""217"");
101                       | lernsequenz auf taste legen ("o", ""218"");
102                       | lernsequenz auf taste legen ("u", ""219"");

```

```
103 | lernsequenz auf taste legen ("k", ""220"");
104 | lernsequenz auf taste legen ("-", ""221"");
105 | lernsequenz auf taste legen ("#", ""222"");
106 | lernsequenz auf taste legen (" ", ""223"");
107 | lernsequenz auf taste legen ("B", ""251"");
108 | lernsequenz auf taste legen ("s", ""251"");
109 | END PROC std tastenbelegung;
110 |
111 |
112 | END PACKET tasten verwaltung;
```

```

1 editorpaket *****|PACKET editor paket DEFINES (* EDITOR
+ |123 *)
2 |(* *****) (* 19.07.85
+ | -bk- *)
3 | (* 10.09.85
+ | -ws- *)
4 | (* 25.04.86
+ | -sh- *)
5 | edit, editget, (* 10.06.86
+ | -wk- *)
6 | quit, quit last, (* 04.06.86
+ | -jl- *)
7 | push, type,
8 | word wrap, margin,
9 | write permission,
10 | set busy indicator,
11 | two bytes,
12 | is kanji esc,
13 | within kanji,
14 | rubin mode,
15 | is editget,
16 | editget command,
17 | getchar, nichts neu,
18 | getcharety, satznr neu,
19 | is incharety, ueberschrift neu,
20 | get window, zeile neu,
21 | get editcursor, abschnitt neu,
22 | get editline, bildabschnitt neu,
23 | put editline, bild neu,
24 | aktueller editor, alles neu,
25 | groesster editor, satznr zeigen,
26 | open editor, ueberschrift zeigen,
27 | editfile, bild zeigen:
28
29
30 |LET hop = "1", right = "2",
31 | up char = "3", clear eop = "4",
32 | clear eol = "5", cursor pos = "6",
33 | piep = "7", left = "8",
34 | down char = "10", rubin = "11",
35 | rubout = "12", cr = "13",
36 | mark key = "16", abschr = "17",
37 | inscr = "18", dezimal = "19",
38 | backcr = "20", esc = "27",
39 | dach = "94", blank = " ",
40
41
42 |LET no output = 0, out zeichen = 1,
43 | out feldrest = 2, out feld = 3,
44 | clear feldrest = 4;
45
46 |LET FELDSTATUS = STRUCT (INT stelle, alte stelle, rand, limit,
47 | anfang, marke, laenge, verschoben,
48 | BOOL einfuegen, fliesstext, write
49 | access,
50 | TEXT tabulator);
51 |FELDSTATUS VAR feldstatus;
52 |TEXT VAR begin mark := "15",
53 | end mark := "14";
54
55 |TEXT VAR separator := "", kommando := "", audit := "", zeichen := "",

```

```

56                                     |      satzrest := "", merksatz := "", alter editsatz := "";
57                                     |
58                                     | INT VAR kommando zeiger := 1, umbruchstelle, umbruch verschoben,
59                                     |      zeile, spalte, output mode := no output, postblanks := @,
60                                     |      min schreibpos, max schreibpos, cpos, absatz ausgleich;
61                                     |
62                                     | BOOL VAR lernmodus := FALSE, separator eingestellt := FALSE,
63                                     |      invertierte darstellung := FALSE, absatzmarke steht,
64                                     |      cursor diff := FALSE, editget modus := FALSE,
65                                     |      two byte mode := FALSE, std fliesstext := TRUE,
66                                     |      editget kommando darf ausgeführt werden := TRUE;.
67                                     |
68 schirmbreite | schirmbreite : x size - 1 .
69 schirmhoehe  | schirmhoehe  : y size .
70 maxbreite   | maxbreite    : schirmbreite - 2 .
71 maxlaenge   | maxlaenge    : schirmhoehe - 1 .
72 marklength  | marklength   : mark size .;
73
74             | initialisiere editor;
75
76             | .initialisiere editor :
77             |      anfang := 1; zeile := @; verschoben := @; tabulator := "";
78             |      einfüegen := FALSE; fliesstext := TRUE; zeileneinfuegen := FALSE;
79             |      marke := @; bildmarke := @; feldmarke := @.;
80
81 editgetcommand ..... | PROC editget command (BOOL CONST schalter) :
82             |      editget kommando darf ausgeführt werden := schalter
83             |      ENDPROC editget command ;
84
85             | (***** editget
86             |      *****)
87
88 editget ..... | PROC editget (TEXT VAR editsatz, INT CONST editlimit, editlaenge,
89             |      TEXT CONST sep, res, TEXT VAR exit char) :
90             |      IF editlaenge < 1 THEN errorstop ("Fenster zu klein") FI;
91             |      separator := ""13""; separator CAT sep;
92             |      separator eingestellt := TRUE;
93             |      TEXT VAR reservierte editget tasten := ""11""12"" ;
94             |      reservierte editget tasten CAT res ;
95             |      disable stop;
96             |      absatz ausgleich := @; exit char := ""; get cursor;
97             |      FELDSTATUS CONST alter feldstatus := feldstatus;
98             |      feldstatus := FELDSTATUS : (1, 1, spalte - 1, editlimit,
99             |      1, @, editlaenge, @,
100            |      FALSE, FALSE, TRUE, "");
101
102            | konstanten neu berechnen;
103            | output mode := out feld;
104            | feld editieren;
105            | zeile verlassen;
106            | feldstatus := alter feldstatus;
107            | konstanten neu berechnen;
108            | separator := "";
109            | separator eingestellt := FALSE .

```

```

108
109      feldeditieren      feld editieren :
110      REP
111      feldeditor (editsatz, reservierte editget tasten);
112      IF is error
113      THEN kommando zeiger := 1; kommando := ""; LEAVE feld editieren
114      FI ;
115      TEXT VAR t, zeichen; getchar (zeichen);
116      IF zeichen ist separator
117      THEN exit char := zeichen; LEAVE feld editieren
118      ELIF zeichen = hop
119      THEN feldout (editsatz, stelle); getchar (zeichen)
120      ELIF zeichen = mark key
121      THEN output mode := out feld
122      ELIF zeichen = abscr
123      THEN exit char := cr; LEAVE feld editieren
124      ELIF zeichen = esc
125      THEN getchar (zeichen); auf exit pruefen;
126      IF zeichen = rubout
127      (*sh*)
128      THEN IF marke > 0
129      THEN merksatz := subtext (editsatz, marke, stelle - 1);
130      change (editsatz, marke, stelle - 1, "");
131      stelle := marke; marke := 0; konstanten neu
132      berechnen
133      FI
134      ELIF zeichen = rubin
135      THEN t := subtext (editsatz, 1, stelle - 1);
136      t CAT merksatz;
137      satzrest := subtext (editsatz, stelle);
138      t CAT satzrest;
139      stelle INCR LENGTH merksatz;
140      merksatz := ""; editsatz := t
141      ELIF editget kommando darf ausgeführt werden
142      CAND
143      zeichen ist kein esc kommando
144      CAND
145      kommando auf taste (zeichen) <> ""
146      THEN editget kommando ausfuehren
147      FI ;
148      output mode := out feld
149      FI
150      PER .
151
152      zeichenistkeineskomma zeichen ist kein esc kommando :
153      +
154      (*wk*)
155      pos (hop + left + right, zeichen) = 0 .
156
157      zeileverlassen      zeile verlassen :
158      IF marke > 0 OR verschoben <> 0
159      THEN stelle DECR verschoben; verschoben := 0; feldout (editsatz, 0);
160      ELSE cursor (rand + 1 + min (LENGTH editsatz, editlaenge), zeile)
161      FI .
162
163      zeichenistseparator zeichen ist separator : pos (separator, zeichen) > 0 .
164

```

```

161 aufexitpruefen | auf exit pruefen :
162 | IF pos (res, zeichen) > 0
163 | THEN exit char := esc + zeichen; LEAVE feld editieren
164 | FI .
165
166 editgetkommandoausfueh | editget kommando ausfuehren :
167 | editget zustaende sichern ;
168 | do (kommando auf taste (zeichen)) ;
169 | alte editget zustaende wieder herstellen ;
170 | IF stelle < marke THEN stelle := marke FI;
171 | konstanten neu berechnen .
172
173 editgetzustaeendesicher | editget zustaeende sichern :
+ | (*wk*)
174 | BOOL VAR alter editget modus := editget modus;
175 | FELDSTATUS VAR feldstatus vor do kommando := feldstatus ;
176 | INT VAR zeile vor do kommando := zeile ;
177 | TEXT VAR separator vor do kommando := separator ;
178 | BOOL VAR separator eingestellt vor do kommando := separator
+ | eingestellt ;
179 | editget modus := TRUE ;
180 | alter editsatz := editsatz .
181
182 alteeditgetzustaeendewi | alte editget zustaeende wieder herstellen :
183 | editget modus := alter editget modus ;
184 | editsatz := alter editsatz;
185 | feldstatus := feldstatus vor do kommando ;
186 | zeile := zeile vor do kommando ;
187 | separator := separator vor do kommando ;
188 | separator eingestellt := separator eingestellt vor do kommando .
189
190 |END PROC editget;
191
192 editget ..... |PROC editget (TEXT VAR editsatz, INT CONST editlimit, TEXT VAR exit
+ | char) :
193 | editget (editsatz, editlimit, x size - x cursor, "", "", exit chr/
194 | END PROC editget; (* 05.07.84
+ | -bk- *)
195
196 editget ..... |PROC editget (TEXT VAR editsatz, TEXT CONST sep, res, TEXT VAR exit
+ | char) :
197 | editget (editsatz, max text length, x size - x cursor, sep, res,
+ | exit char)
198 | END PROC editget; (* 05.07.84
+ | -bk- *)
199
200 editget ..... |PROC editget (TEXT VAR editsatz) :
201 | TEXT VAR exit char; (* 05.07.84/
+ | -bk- *)
202 | editget (editsatz, max text length, x size - x cursor, "", "",
+ | exit char)
203 | END PROC editget;
204

```

```

205 editget .....|PROC editget (TEXT VAR editsatz, INT CONST editlimit, editlaenge) :
206 |   TEXT VAR exit char;
207 |   editget (editsatz, editlimit, editlaenge, "", "", exit char)
208 |ENDPROC editget;
209
210 |((***** feldeditor
+ |*****))
211
212 |TEXT VAR reservierte feldeditor tasten ;
+ |   (*jl*)
213
214 feldeditor .....|PROC feldeditor (TEXT VAR satz, TEXT CONST res) :
215 |   enable stop;
216 |   reservierte feldeditor tasten := "1"2"8" ;
217 |   reservierte feldeditor tasten CAT res;
218 |   absatzmarke steht := (satz SUB LENGTH satz) = blank;
219 |   alte stelle merken;
220 |   cursor diff bestimmen und ggf ausgleichen;
221 |   feld editieren;
222 |   absatzmarke updaten .
223
224 altestellemerken |alte stelle merken : alte stelle := stelle .
225
226 cursordiffbestimmenund |cursor diff bestimmen und ggf ausgleichen :
227 |   IF cursor diff
228 |   THEN stelle INCR 1; cursor diff := FALSE
229 |   FI ;
230 |   IF stelle auf zweitem halbzeichen
231 |   THEN stelle DECR 1; cursor diff := TRUE
232 |   FI .
233
234 feldeditieren |feld editieren :
235 |   REP
236 |   feld optisch aufbereiten;
237 |   kommando annehmen und ausfuehren
238 |   PER .
239
240 absatzmarkeupdaten |absatzmarke updaten :
241 |   IF absatzmarke soll stehen
242 |   THEN IF NOT absatzmarke steht THEN absatzmarke schreiben (TRUE) FI
243 |   ELSE IF absatzmarke steht THEN absatzmarke schreiben (FALSE) FI
244 |   FI .
245
246 absatzmarkesollstehen |absatzmarke soll stehen : (satz SUB LENGTH satz) = blank .
247
248 feldoptischaufbereiten |feld optisch aufbereiten :
249 |   stelle korrigieren;
250 |   verschieben wenn erforderlich;
251 |   randausgleich fuer doppelzeichen;
252 |   output mode behandeln;
253 |   ausgabe verhindern .
254

```

```

255 randausgleichfuerdoppe | randausgleich fuer doppelzeichen :
256 | IF stelle = max schreibpos CAND stelle auf erstem halbzeichen
257 | THEN verschiebe (1)
258 | FI .
259
260 stellekorrigieren | stelle korrigieren :
261 | IF stelle auf zweitem halbzeichen THEN stelle DECR 1 FI .
262
263 stelleaufferstemhalbzei | stelle auf erstem halbzeichen : within kanji (satz, stelle + 1) .
264
265 stelleaufzweitemhalbze | stelle auf zweitem halbzeichen : within kanji (satz, stelle) .
266
267 outputmodebehandeln | output mode behandeln :
268 | SELECT output mode OF
269 | CASE no output : im markiermode markierung anpassen
270 | CASE out zeichen : zeichen ausgeben; LEAVE output mode
+ | behandeln
271 | CASE out feldrest : feldrest neu schreiben
272 | CASE out feld : feldout (satz, stelle)
273 | CASE clear feldrest : feldrest loeschen
274 | END SELECT;
275 | schreibmarke positionieren (stelle) .
276
277 ausgabeverhindern | ausgabe verhindern : output mode := no output .
278
279 immarkiermodemarkierun | im markiermode markierung anpassen :
280 | IF markiert THEN markierung anpassen FI .
281
282 markierunganpassen | markierung anpassen :
283 | IF stelle > alte stelle
284 | THEN markierung verlaengern
285 | ELIF stelle < alte stelle
286 | THEN markierung verkuerzen
287 | FI .
288
289 markierungverlaengern | markierung verlaengern :
290 | invers out (satz, alte stelle, stelle, "", end mark) .
291
292 markierungverkuerzen | markierung verkuerzen :
293 | invers out (satz, stelle, alte stelle, end mark, "") .
294
295 zeichenausgeben | zeichen ausgeben :
296 | IF NOT markiert
297 | THEN out (zeichen)
298 | ELIF mark refresh line mode
299 | THEN feldout (satz, stelle); schreibmarke positionieren (stelle)
300 | ELSE out (begin mark); markleft; out (zeichen); out (end mark);
+ | markleft
301 | FI .
302

```

```

303 markleft |markleft :
304 | marklength TIMESOUT left .
305
306 feldrestneuschreiben |feldrest neu schreiben :
307 | IF NOT markiert
308 | THEN feldrest unmarkiert neu schreiben
309 | ELSE feldrest markiert neu schreiben
310 | FI ;
311 | WHILE postblanks > @ CAND x cursor <= rand + laenge REP
312 | out (blank); postblanks DECR 1
313 | PER ; postblanks := @ .
314
315 feldrestunmarkiertneus |feldrest unmarkiert neu schreiben :
316 | schreibmarke positionieren (alte stelle);
317 | out subtext mit randbehandlung (satz, alte stelle, stelle am ende)
+
318
319 feldrestmarkiertneusch |feldrest markiert neu schreiben :
320 | markierung verlaengern; out subtext mit randbehandlung
321 | (satz, stelle, stelle am ende - 2 *
+
322 | marklength) .
323
323 kommandoannehmenundaus |kommando annehmen und ausfuehren :
324 | kommando annehmen; kommando ausfuehren .
325
326 kommandoannehmen |kommando annehmen :
327 | getchar (zeichen); kommando zurueckweisen falls noetig .
328
329 kommandozurueckweisenf |kommando zurueckweisen falls noetig :
330 | IF NOT write access CAND zeichen ist druckbar
331 | THEN benutzer warnen; kommando ignorieren
332 | FI .
333
334 benutzerwarnen |benutzer warnen : out (piep) .
335
336 kommandoignorieren |kommando ignorieren :
337 | zeichen := ""; LEAVE kommando annehmen und ausfuehren .
338
339 kommandoausfuehren |kommando ausfuehren :
340 | neue satzlaenge bestimmen;
341 | alte stelle merken;
342 | IF zeichen ist separator
343 | THEN feldeditor verlassen
344 | ELIF zeichen ist druckbar
345 | THEN fortschreiben
346 | ELSE funktionstasten behandeln
347 | FI .
348
349 neuesatzlaengebestimme |neue satzlaenge bestimmen : INT VAR satzlaenge := LENGTH satz .
350

```

351	feldeditorverlassen	feldeditor verlassen :
352	+	IF NOT absatzmarke steht THEN blanks abschneiden FI;
353		(*sh*)
354		push (zeichen); LEAVE feld editieren .
355	blanksabschneiden	blanks abschneiden :
356		INT VAR letzte non blank pos := satzlaenge;
357		WHILE letzte non blank pos > 0 CAND (satz SUB letzte non blank
358	+	pos) = blank REP
359		letzte non blank pos DECR 1
360		PER; satz := subtext (satz, 1, letzte non blank pos) .
361	zeichenistdruckbar	zeichen ist druckbar : zeichen >= blank .
362		
363	zeichenistseparator	zeichen ist separator :
364		separator eingestellt CAND pos (separator, zeichen) > 0 .
365		
366	fortschreiben	fortschreiben :
367		zeichen in satz eintragen;
368		IF is kanji esc (zeichen) THEN kanji zeichen schreiben FI;
369		bei erreichen von limit ueberlauf behandeln .
370		
371	zeicheninsatzeinragen	zeichen in satz eintragen :
372		IF hinter dem satz
373		THEN satz mit leerzeichen auffuellen und zeichen anfüegen
374		ELIF einfuegen
375		THEN zeichen vor aktueller position einfuegen
376		ELSE altes zeichen ersetzen
377		FI .
378		
379	hinterdemsatz	hinter dem satz : stelle > satzlaenge .
380		
381	satzmitleerzeichenauff	satz mit leerzeichen auffuellen und zeichen anfüegen :
382		satz AUFFUELLENMIT blank;
383		zeichen anfüegen;
384		output mode := out zeichen .
385		
386	zeichenanfüegen	zeichen anfüegen : satz CAT zeichen; neue satzlaenge bestimmen .
387	zeichenignorieren	zeichen ignorieren : benutzer warnen; LEAVE kommando ausführen .
388		
389	zeichenvoraktuellerpos	zeichen vor aktueller position einfuegen :
390		insert char (satz, zeichen, stelle);
391		neue satzlaenge bestimmen;
392		output mode := out feldrest .
393		
394	alteszeichenersetzen	altes zeichen ersetzen :
395		replace (satz, stelle, zeichen);
396		IF stelle auf erstem halbzeichen
397		THEN output mode := out feldrest; replace (satz, stelle + 1, blank)
398		ELSE output mode := out zeichen
399		FI .

```

400
401   kanjizeichenschreiben | kanji zeichen schreiben :
402                       | alte stelle merken;
403                       | stelle INCR 1; getchar (zeichen);
404                       | IF zeichen < ""64"" THEN zeichen := ""64"" FI;
405                       | IF hinter dem satz
406                       | THEN zeichen anfüegen
407                       | ELIF einfuegen
408                       | THEN zeichen vor aktueller position einfuegen
409                       | ELSE replace (satz, stelle, zeichen)
410                       | FI ;
411                       | output mode := out feldrest .
412
413   beierreichenvonlimitue | bei erreichen von limit ueberlauf behandeln :
+                       | (*sh*)
414                       | IF satzlaenge kritisch
415                       | THEN in naechste zeile falls moeglich
416                       | ELSE stelle INCR 1
417                       | FI .
418
419   satzlaengekritisch    | satzlaenge kritisch :
420                       | IF stelle >= satzlaenge
421                       | THEN satzlaenge = limit
422                       | ELSE satzlaenge = limit + 1
423                       | FI .
424
425   innaechstezeilefallsmo | in naechste zeile falls moeglich :
426                       | IF fliesstext AND umbruch moeglich OR NOT fliesstext AND stelle
+                       | >= satzlaenge
427                       | THEN in naechste zeile
428                       | ELSE stelle INCR 1
429                       | FI .
430
431   umbruchmoeglich      | umbruch moeglich :
432                       | INT CONST st := stelle; stelle := limit;
433                       | INT CONST ltzt wortanf := letzter wortanfang (satz);
434                       | stelle := st; einrueckposition (satz) < ltzt wortanf .
435
436   innaechstezeile      | in naechste zeile :
437                       | IF fliesstext
438                       | THEN ueberlauf und oder umbruch
439                       | ELSE ueberlauf ohne umbruch
440                       | FI .
441
442   ueberlaufundoderumbruc | ueberlauf und oder umbruch :
443                       | INT VAR umbruchpos := 1;
444                       | umbruchposition bestimmen;
445                       | loeschposition bestimmen;
446                       | IF stelle = satzlaenge
447                       | THEN ueberlauf mit oder ohne umbruch
448                       | ELSE umbruch mit oder ohne ueberlauf
449                       | FI .
450

```

```

451 umbruchpositionbestimm | umbruchposition bestimmen :
452 | umbruchstelle := stelle;
453 | stelle := satzlaenge;
454 | umbruchpos := max (umbruchpos, letzter wortanfang (satz));
455 | stelle := umbruchstelle .
456
457 loeschpositionbestimme | loeschposition bestimmen :
458 | INT VAR loeschpos := umbruchpos;
459 | WHILE davor noch blank REP loeschpos DECR 1 PER .
460
461 davornochblank | davor noch blank :
462 | loeschpos > ganz links CAND (satz SUB (loeschpos - 1)) = blank .
463
464 ganzlinks | ganz links : max (1, marke) .
465
466 ueberlaufmitoderohneum | ueberlauf mit oder ohne umbruch :
467 | IF zeichen = blank OR loeschpos = ganz links
468 | THEN stelle := 1; ueberlauf ohne umbruch
469 | ELSE ueberlauf mit umbruch
470 | FI .
471
472 ueberlaufohneumbruch | ueberlauf ohne umbruch : push (cr) .
473
474 ueberlaufmitumbruch | ueberlauf mit umbruch :
475 | ausgabe verhindern;
476 | umbruchkommando aufbereiten;
477 | auf loeschposition positionieren .
478
479 umbruchkommandoaufbere | umbruchkommando aufbereiten :
480 | zeichen := hop + rubout + inscr;
481 | satzrest := subtext (satz, umbruchpos);
482 | zeichen CAT satzrest;
483 | IF stelle ist im umgebrochenen teil
484 | THEN insert char (zeichen, backcr, max (stelle - umbruchpos + 1,
+ | 0) + 4);
485 | zeichen CAT backcr
486 | FI ;
487 | push (zeichen) .
488
489 stelleistimungebrochen | stelle ist im umgebrochenen teil : stelle >= loeschpos .
490
491 aufloeschpositionposit | auf loeschposition positionieren : stelle := loeschpos .
492
493 umbruchmitoderohneuebe | umbruch mit oder ohne ueberlauf :
494 | umbruchposition anpassen;
495 | IF stelle ist im umgebrochenen teil
496 | THEN umbruch mit ueberlauf
497 | ELSE umbruch ohne ueberlauf
498 | FI .
499

```

```

500 umbruchpositionanpasse | umbruchposition anpassen :
501 | IF zeichen = blank
502 | THEN umbruchpos := stelle + 1;
503 | umbruchposition bestimmen;
504 | neue loeschposition bestimmen
505 | FI .
506 |
507 neuuloeschpositionbest | neue loeschposition bestimmen :
508 | loeschpos := umbruchpos;
509 | WHILE davor noch blank AND stelle noch nicht erreicht REP
+ | loeschpos DECR 1 PER .
510 |
511 stellenochnichterreicht | stelle noch nicht erreicht : loeschpos > stelle + 1 .
512 |
513 umbruchmitueberlauf | umbruch mit ueberlauf : ueberlauf mit umbruch .
514 |
515 umbruchohneueberlauf | umbruch ohne ueberlauf :
516 | zeichen := inscr;
517 | satzrest := subtext (satz, umbruchpos);
518 | zeichen CAT satzrest;
519 | zeichen CAT up char + backcr;
520 | umbruchstelle INCR 1; umbruch verschoben := verschoben;
521 | satz := subtext (satz, 1, loeschpos - 1);
522 | schreibmarke positionieren (loeschpos); feldrest loeschen;
523 | output mode := out feldrest;
524 | push (zeichen) .
525 |
526 funktionstastenbehandle | funktionstasten behandeln :
527 | SELECT pos (kommandos, zeichen) OF
528 | CASE c hop : hop kommandos behandeln
529 | CASE c esc : esc kommandos behandeln
530 | CASE c right : nach rechts oder ueberlauf
531 | CASE c left : wenn moeglich ein schritt nach links
532 | CASE c tab : zur naechsten tabulator position
533 | CASE c dezimal : dezimalen schreiben
534 | CASE c rubin : einfuegen umschalten
535 | CASE c rubout : ein zeichen loeschen
536 | CASE c abscr, c inscr, c down : feldeditor verlassen
537 | CASE c up : eine zeile nach oben
+ | (*sh*)
538 | CASE c cr : ggf absatz erzeugen
539 | CASE c mark : markieren umschalten
540 | CASE c backcr : zurueck zur umbruchstelle
541 | OTHERWISE : sondertaste behandeln
542 | END SELECT .
543 |
544 kommandos | kommandos :
545 | LET c hop = 1, c right = 2,
546 | c up = 3, c left = 4,
547 | c tab = 5, c down = 6,
548 | c rubin = 7, c rubout = 8,
549 | c cr = 9, c mark = 10,
550 | c abscr = 11, c inscr = 12,
551 | c dezimal = 13, c esc = 14,
552 | c backcr = 15;
553 |

```

```

554      ""1""2""3""8""9""10""11""12""13""16""17""18""19""27""20"" .
555
556      dezimalenschreiben | dezimalen schreiben : IF write access THEN dezimaleditor (satz) FI .
557
558      zurueckzurumbruchstell | zurueck zur umbruchstelle:
559      IF umbruch stelle > @ THEN stelle := umbruch stelle FI;
560      IF verschoben <> umbruch verschoben
561      THEN verschoben := umbruch verschoben; output mode := out feld
562      FI .
563
564      hopkommandosbehandeln | hop kommandos behandeln :
565      TEXT VAR zweites zeichen; getchar (zweites zeichen);
566      zeichen CAT zweites zeichen;
567      SELECT pos (hop kommandos, zweites zeichen) OF
568      CASE h hop      : nach links oben
569      CASE h right   : nach rechts blaettern
570      CASE h left    : nach links blaettern
571      CASE h tab     : tab position definieren oder loeschen
572      CASE h rubin   : zeile splitten
573      CASE h rubout  : loeschen oder rekombinieren
574      CASE h cr, h up, h down : feldeditor verlassen
575      OTHERWISE     : zeichen ignorieren
576      END SELECT .
577
578      hopkommandos | hop kommandos :
579      LET h hop      = 1,          h right = 2,
580      h up          = 3,          h left  = 4,
581      h tab         = 5,          h down  = 6,
582      h rubin       = 7,          h rubout = 8,
583      h cr          = 9;
584
585      ""1""2""3""8""9""10""11""12""13"" .
586
587      nachlinksoben | nach links oben :
588      stelle := max (marke, anfang) + verschoben; feldeditor verlassen
589
590      nachrechtsblaettern | nach rechts blaettern :
591      INT CONST rechter rand := stelle am ende - markierausgleich;
592      IF stelle ist am rechten rand
593      THEN stelle INCR laenge - 2 * markierausgleich + ausgleich fuer
594      doppelzeichen
595      ELSE stelle := rechter rand
596      FI ;
597      IF satzlaenge <= limit THEN stelle := min (stelle, limit) FI;
598      alte einrueckposition mitziehen .
599
600      stelleistamrechtenrand | stelle ist am rechten rand :
601      stelle auf erstem halbzeichen CAND stelle = rechter rand - 1
602      COR stelle = rechter rand .
603
604      ausgleichfuerdoppelzei | ausgleich fuer doppelzeichen : stelle - rechter rand .

```

```

605 nachlinksblaettern nach links blaettern :
606 INT CONST linker rand := stelle am anfang;
607 IF stelle = linker rand
608 THEN stelle DECR laenge - 2 * markierausgleich
609 ELSE stelle := linker rand
610 FI ;
611 stelle := max (ganz links, stelle);
612 alte einrueckposition mitziehen .
613

614 tabpositiondefinieren tab position definieren oder loeschen :
615 IF stelle > LENGTH tabulator
616 THEN tabulator AUFFUELLENMIT right; tabulator CAT dach
617 ELSE replace (tabulator, stelle, neues tab zeichen)
618 FI ;
619 feldeditor verlassen .
620

621 neuestabzeichen neues tab zeichen :
622 IF (tabulator SUB stelle) = right THEN dach ELSE right FI .
623

624 zeilesplitten zeile splitten :
625 IF write access THEN feldeditor verlassen ELSE zeichen ignorieren
+ FI .
626

627 loeschenoderrekombinie loeschen oder rekombinieren :
628 IF NOT write access
629 THEN zeichen ignorieren
630 ELIF hinter dem satz
631 THEN zeilen rekombinieren
632 ELIF auf erstem zeichen
633 THEN ganze zeile loeschen
634 ELSE zeilenrest loeschen
635 FI .
636

637 zeilenrekombinieren zeilen rekombinieren : feldeditor verlassen .

638 auferstemzeichen auf erstem zeichen : stelle = 1 .

639 ganzezeileloeschen ganze zeile loeschen : satz := ""; feldeditor verlassen .
640

641 zeilenrestloeschen zeilenrest loeschen :
642 change (satz, stelle, satzlaenge, "");
643 output mode := clear feldrest .
644

645 esckommandosbehandeln esc kommandos behandeln :
646 getchar (zweites zeichen);
647 zeichen CAT zweites zeichen;
648 auf exit pruefen;
649 SELECT pos (esc kommandos, zweites zeichen) OF
650 CASE e hop : lernmodus umschalten
651 CASE e right : zum naechsten wort
652 CASE e left : zum vorigen wort
653 OTHERWISE : belegte taste ausfuehren
654 END SELECT .
655

```

```

656 aufexitpruefen |auf exit pruefen :
657 | IF pos (res, zweites zeichen) > 0 THEN feldeditor verlassen FI .
658 |
659 esckommandos |esc kommandos :
660 | LET e hop = 1,
661 | e right = 2,
662 | e left = 3;
663 |
664 | ""1""2""8"" .
665 |
666 lernmodusumschalten |lernmodus umschalten :
667 | IF lernmodus THEN lernmodus ausschalten ELSE lernmodus einschalten
+ | FI;
668 | feldeditor verlassen .
669 |
670 lernmodusausschalten |lernmodus ausschalten :
671 | lernmodus := FALSE;
672 | belegbare taste erfragen;
673 | audit := subtext (audit, 1, LENGTH audit - 2);
674 | IF taste = hop
675 | THEN (* lernsequenz nicht auf taste legen *) (* 16.08.85
+ | -ws- *)
676 | ELSE lernsequenz auf taste legen (taste, audit)
677 | FI ;
678 | audit := "" .
679 |
680 belegbaretasteerfragen |belegbare taste erfragen :
681 | TEXT VAR taste; getchar (taste);
682 | WHILE taste ist reserviert REP
683 | benutzer warnen; getchar (taste)
684 | PER .
685 |
686 tasteistreserviert |taste ist reserviert : (* 16.08.85
+ | -ws- *)
687 | taste <> hop CAND pos (reservierte feldeditor tasten, taste) > 0 .
688 |
689 lernmoduseinschalten |lernmodus einschalten : audit := ""; lernmodus := TRUE .
690 |
691 zumvorigenwort |zum vorigen wort :
692 | IF stelle > 1
693 | THEN stelle DECR 1; stelle := letzter wortanfang (satz);
694 | alte einrueckposition mitziehen;
695 | IF (satz SUB stelle) <> blank THEN LEAVE zum vorigen wort FI
696 | FI ;
697 | feldeditor verlassen .
698 |
699 zumnaechstenwort |zum naechsten wort :
700 | IF kein naechstes wort THEN feldeditor verlassen FI .
701 |
702 keinnaechsteswort |kein naechstes wort :
703 | BOOL VAR im alten wort := TRUE;
704 | INT VAR i;
705 | FOR i FROM stelle UPTO satzlaenge REP

```

```

706 |         IF im alten wort
707 |         THEN im alten wort := (satz SUB i) <> blank
708 |         ELIF (satz SUB i) <> blank
709 |         THEN stelle := i; LEAVE kein naechstes wort WITH FALSE
710 |         FI
711 |     PER;
712 |     TRUE .
713 |
714 | belegtetastenausfuehren | belegte taste ausfuehren :
715 | |         IF ist kommando taste
716 | |         THEN feldditor verlassen
717 | |         ELSE gelerntes ausfuehren
718 | |         FI .
719 |
720 | istkommandotaste | ist kommando taste : taste enthaelt kommando (zweites zeichen) .
721 |
722 | gelerntesausfuehren | gelerntes ausfuehren :
723 | |     push (lernsequenz auf taste (zweites zeichen)) .
724 | |     (*sh*)
725 |
726 | nachrechtsoderueberlau | nach rechts oder ueberlauf :
727 | |     IF fliesstext COR stelle < limit OR satzlaenge > limit
728 | |     THEN nach rechts
729 | |     ELSE auf anfang der naechsten zeile
730 | |     FI .
731 |
732 | nachrechts | nach rechts :
733 | |     IF stelle auf erstem halbzeichen THEN stelle INCR 2 ELSE stelle
734 | |     INCR 1 FI;
735 | |     alte einrueckposition mitziehen .
736 |
737 | aufanfangdernaechstenz | auf anfang der naechsten zeile : push (abscr) .
738 |
739 | nachlinks | nach links : stelle DECR 1; alte einrueckposition mitziehen .
740 |
741 | alteeinrueckpositionmi | alte einrueckposition mitziehen :
742 | |     IF satz ist leerzeile
743 | |     THEN alte einrueckposition := stelle
744 | |     ELSE alte einrueckposition := min (stelle, einrueckposition (satz))
745 | |     FI .
746 |
747 | satzistleerzeile | satz ist leerzeile :
748 | |     satz = "" OR satz = blank .
749 |
750 | wennmoeglicheinschritt | wenn moeglich ein schritt nach links :
751 | |     IF stelle = ganz links
752 | |     THEN zeichen ignorieren
753 | |     ELSE nach links
754 | |     FI .

```

```

754 zurnaechstentabulatorp |zur naechsten tabulator position :
755 |bestimme naechste explizite tabulator position;
756 |IF tabulator gefunden
757 |THEN explizit tabulieren
758 |ELIF stelle <= satzlaenge
759 |THEN implizit tabulieren
760 |ELSE auf anfang der naechsten zeile
761 |FI .
762
763 bestimmenaechsteexpliz |bestimme naechste explizite tabulator position :
764 |INT VAR tab position := pos (tabulator, dach, stelle + 1);
765 |IF tab position > limit AND satzlaenge <= limit
766 |THEN tab position := 0
767 |FI .
768
769 tabulatorgefunden |tabulator gefunden : tab position <> 0 .
770
771 explizittabulieren |explizit tabulieren : stelle := tab position; push (dezimal) .
772
773 implizittabulieren |implizit tabulieren :
774 |tab position := einrueckposition (satz);
775 |IF stelle < tab position
776 |THEN stelle := tab position
777 |ELSE stelle := satzlaenge + 1
778 |FI .
779
780 einfuegenumschalten |einfuegen umschalten :
781 |IF NOT write access THEN zeichen ignorieren FI;
782 |(*sh*)
783 |einfuegen := NOT einfuegen;
784 |IF einfuegen THEN einfuegen optisch anzeigen FI;
785 |feldeditor verlassen .
786
787 einfuegenoptischanzeig |einfuegen optisch anzeigen :
788 |IF markiert
789 |THEN out (begin mark); markleft; out (dach left); warten;
790 |out (end mark); markleft
791 |ELSE out (dach left); warten;
792 |IF stelle auf erstem halbzeichen
793 |THEN out text (satz, stelle, stelle + 1)
794 |ELSE out text (satz, stelle, stelle)
795 |FI
796 |FI .
797
797 markiert |markiert : marke > 0 .
798
798 dachleft |dach left : ""94""8"" .
799
800 warten |warten :
801 |TEXT VAR t := incharety (2);
802 |kommando CAT t; IF lernmodus THEN audit CAT t FI .
803

```

```

804 einzeichenloeschen |ein zeichen loeschen :
805 | IF NOT write access THEN zeichen ignorieren FI;
+ | (*sh*)
806 | IF zeichen davor soll geloescht werden
807 | THEN nach links oder ignorieren
808 | FI ;
809 | IF NOT hinter dem satz THEN aktuelles zeichen loeschen FI .
810
811 zeichendavorsollgeloes |zeichen davor soll geloescht werden :
812 | hinter dem satz COR markiert .
813
814 nachlinksoderignoriere |nach links oder ignorieren :
815 | IF stelle > ganz links
816 | THEN nach links
+ | (*sh*)
817 | ELSE zeichen ignorieren
818 | FI .
819
820 aktuelleszeichenloesch |aktuelles zeichen loeschen :
821 | stelle korrigieren; alte stelle merken;
822 | IF stelle auf erstem halbzeichen
823 | THEN delete char (satz, stelle);
824 | postblanks INCR 1
825 | FI ;
826 | delete char (satz, stelle);
827 | postblanks INCR 1;
828 | neue satzlaenge bestimmen;
829 | output mode := out feldrest .
830
831 einezeilenachoben |eine zeile nach oben :
+ | (*sh*)
832 | IF NOT absatzmarke steht CAND NOT ist teil eines
+ | umbruchkommandos
833 | THEN blanks abschneiden
834 | FI ;
835 | push (zeichen); LEAVE feld editieren .
836
837 istteileinesumbruchkom |ist teil eines umbruchkommandos : (kommando SUB kommandozeiger) =
+ | backcr .
838
839 ggfabsatzerzeugen |ggf absatz erzeugen :
+ | (*sh*)
840 | IF write access
841 | THEN IF NOT absatzmarke steht THEN blanks abschneiden FI;
842 | IF stelle > LENGTH satz AND fliesstext AND (satz SUB LENGTH
+ | satz) <> blank
843 | THEN satz CAT blank
844 | FI
845 | FI ; push (zeichen); LEAVE feld editieren .
846
847 markierenumschalten |markieren umschalten :
848 | IF markiert
849 | THEN marke := 0; maxschreibpos INCR marklength; cpos DECR
+ | marklength
850 | ELSE marke := stelle; maxschreibpos DECR marklength; cpos INCR
+ | marklength;

```

```

851      |      verschieben wenn erforderlich
852      |      FI ;
853      |      feldeditor verlassen .
854      |
855      |sondertaste behandeln : push (esc + zeichen) .
856      |END PROC feldeditor;
857      |
858      |PROC dezimaleditor (TEXT VAR satz) :
859      |  INT VAR dezimalanfang := stelle;
860      |  zeichen einlesen;
861      |  IF dezimalstartzeichen CAND ueberschreibbar THEN dezimalen
862      |    schreiben FI;
863      |  push (zeichen) .
864      |
864      | zeicheneinlesen      | zeichen einlesen      : TEXT VAR zeichen; getchar (zeichen) .
865      | dezimalzeichen      | dezimalzeichen       : pos (dezimalen, zeichen) > 0 AND nicht
866      |      +                |      separator .
866      | dezimalstartzeichen  | dezimalstartzeichen  : pos (startdezimalen, zeichen) > 0 AND nicht
867      |      +                |      separator .
867      | dezimalen            | dezimalen             : "0123456789" .
868      | startdezimalen      | startdezimalen       : "+-0123456789" .
869      | nichtseparator      | nicht separator      : pos (separator, zeichen) = 0 .
870      |
871      | ueberschreibbar     | ueberschreibbar      :
872      |      +                |      dezimalanfang > LENGTH satz OR
873      |      +                |      pos (ueberschreibbare zeichen, satz SUB dezimalanfang) > 0 .
874      |
875      | ueberschreibbarezei | ueberschreibbare zeichen : " ,+-0123456789" .
876      |
877      | dezimalenschreiben  | dezimalen schreiben  :
878      |      +                |      REP
879      |      +                |      dezimale in satz eintragen;
880      |      +                |      dezimalen zeigen;
881      |      +                |      zeichen einlesen;
882      |      +                |      dezimalanfang DECR 1
883      |      +                |      UNTIL dezimaleditor beendet PER;
884      |      +                |      stelle INCR 1 .
885      |
886      | dezimaleinsatzeintra | dezimale in satz eintragen :
887      |      +                |      IF dezimalanfang > LENGTH satz
888      |      +                |      THEN satz AUFFUELLENMIT blank; satz CAT zeichen
889      |      +                |      ELSE delete char (satz, dezimalanfang); insert char (satz,
890      |      +                |      zeichen, stelle)
891      |      +                |      FI .
892      | dezimalenzeigen     | dezimalen zeigen    :
893      |      +                |      INT VAR min dezimalschreibpos := max (min schreibpos,
894      |      +                |      dezimalanfang);

```

```

894      | IF markiert THEN markiert zeigen ELSE unmarkiert zeigen FI;
895      | schreibmarke positionieren (stelle) .
896      |
897      | markiert      | markiert : marke > 0 .
898      |
899      | markiertzeigen | markiert zeigen :
900      |               |   invers out (satz, min dezimalschreibpos, stelle, "", end mark);
901      |               |   out (zeichen) .
902      |
903      | unmarkiertzeigen | unmarkiert zeigen :
904      |               |   schreibmarke positionieren (min dezimalschreibpos);
905      |               |   out subtext (satz, min dezimalschreibpos, stelle) .
906      |
907      | dezimaleditorbeendet | dezimaleditor beendet :
908      |               |   NOT dezimalzeichen OR
909      |               |   dezimalanfang < max (min schreibpos, marke) OR
910      |               |   NOT ueberschreibbar .
911      |               | END PROC dezimaleditor;
912      |
913      | iseditget ..... | BOOL PROC is editget :
914      |               |   editget modus
915      |               | END PROC is editget ;
916      |
917      | geteditline ..... | PROC get editline (TEXT VAR editline, INT VAR editpos, editmarke) :
918      |               |   IF editget modus
919      |               |   THEN editline := alter editsatz;
920      |               |   editpos := stelle
921      |               |   FI ;
922      |               |   editmarke := marke
923      |               | END PROC get editline;
924      |
925      | puteditline ..... | PROC put editline (TEXT CONST editline, INT CONST editpos,
926      |               |   editmarke) :
927      |               |   IF editget modus
928      |               |   THEN alter editsatz := editline;
929      |               |   stelle := max (editpos, 1);
930      |               |   marke := max (editmarke, 0)
931      |               |   FI
932      |               | END PROC put editline;
933      |
934      | withinkanji ..... | BOOL PROC within kanji (TEXT CONST satz, INT CONST stelle) :
935      |               |   count directly prefixing kanji esc bytes;
936      |               |   number of kanji esc bytes is odd .
937      |
938      | countdirectlyprefixing | count directly prefixing kanji esc bytes :
939      |               |   INT VAR pos := stelle - 1, kanji esc bytes := 0;
940      |               |   WHILE pos > 0 CAND is kanji esc (satz SUB pos) REP
941      |               |   kanji esc bytes INCR 1; pos DECR 1
942      |               | PER .

```

```

942
943   numberofkanjiesbytesi | number of kanji esc bytes is odd :
944                         | (kanji esc bytes AND 1) <> 0 .
945                         | END PROC within kanji;
946
947   iskanjiesc ..... | BOOL PROC is kanji esc (TEXT CONST char) :
+                           | (*sh*)
948                         | two byte mode CAND
949                         | (char >= "129" AND char <= "159" OR char >= "224" AND char
+                           | <= "239")
950                         | END PROC is kanji esc;
951
952   twobytes ..... | BOOL PROC two bytes : two byte mode END PROC two bytes;
953
954   twobytes ..... | PROC two bytes (BOOL CONST new mode) :
955                         | two byte mode := new mode
956                         | END PROC two bytes;
957
958   outtext ..... | PROC outtext (TEXT CONST source, INT CONST from, to) :
959                         | out subtext mit randbehandlung (source, from, to);
960                         | INT VAR trailing;
961                         | IF from <= LENGTH source
962                         | THEN trailing := to - LENGTH source
963                         | ELSE trailing := to - from + 1
964                         | FI ; trailing TIMESOUT blank
965                         | END PROC outtext;
966
967   outsubtextmitrandbehan ... | PROC out subtext mit randbehandlung (TEXT CONST satz, INT CONST von,
+                           | bis) :
968                         | IF von > bis
969                         | THEN
970                         | ELIF bis >= LENGTH satz COR NOT within kanji (satz, bis + 1)
971                         | THEN out subtext mit anfangsbehandlung (satz, von, bis)
972                         | ELSE out subtext mit anfangsbehandlung (satz, von, bis - 1); out
+                           | (blank)
973                         | FI
974                         | END PROC out subtext mit randbehandlung;
975
976   outsubtextmitanfangsbe ... | PROC out subtext mit anfangsbehandlung (TEXT CONST satz, INT CONST
+                           | von, bis) :
977                         | IF von > bis
978                         | THEN
979                         | ELIF von = 1 COR NOT within kanji (satz, von)
980                         | THEN out subtext (satz, von, bis)
981                         | ELSE out (blank); out subtext (satz, von + 1, bis)
982                         | FI
983                         | END PROC out subtext mit anfangsbehandlung;
984

```

```

985 getcursor .....|PROC get cursor : get cursor (spalte, zeile) END PROC get
+                  | cursor;
986                  |
987 xcursor .....|INT PROC x cursor : get cursor; spalte END PROC x
+                  | cursor;
988                  |
989 writepermission .....|BOOL PROC write permission : write access END PROC write
+                  | permission;
990                  |
991 push .....|PROC push (TEXT CONST ausfuehrkommando) :
992 | IF ausfuehrkommando = ""
+ | (*sh*)
993 | THEN
994 | ELIF kommando = ""
995 | THEN kommando := ausfuehrkommando
996 | ELIF (kommando SUB kommando zeiger - 1) = ausfuehrkommando
997 | THEN kommando zeiger DECR 1
998 | ELIF replace moeglich
999 | THEN kommando zeiger DECR laenge des ausfuehrkommandos;
1000 | replace (kommando, kommando zeiger, ausfuehrkommando)
1001 | ELSE insert char (kommando, ausfuehrkommando, kommando zeiger)
1002 | FI .
1003 |
1004 replacemoeglich |replace moeglich :
1005 | INT CONST laenge des ausfuehrkommandos := LENGTH ausfuehrkommando;
1006 | kommando zeiger > laenge des ausfuehrkommandos .
1007 |END PROC push;
1008 |
1009 type .....|PROC type (TEXT CONST ausfuehrkommando) :
1010 | kommando CAT ausfuehrkommando
1011 |END PROC type;
1012 |
1013 stelleamfang .....|INT PROC stelle am anfang : anfang + verschoben END PROC stelle am
+ | anfang;
1014 |
1015 stelleamende .....|INT PROC stelle am ende : stelle am anfang+laenge-1 END PROC
+ | stelle am ende;
1016 |
1017 markierausgleich .....|INT PROC markierausgleich : SIGN marke * marklength END PROC
+ | markierausgleich;
1018 |
1019 verschiebenwennerforde ...|PROC verschieben wenn erforderlich :
1020 | IF stelle > max schreibpos
1021 | THEN verschiebe (stelle - max schreibpos)
1022 | ELIF stelle < min schreibpos

```

```

1023 | THEN verschiebe (stelle - min schreibpos)
1024 | FI
1025 | END PROC verschieben wenn erforderlich;
1026 |

1027 verschiebe ..... PROC verschiebe (INT CONST i) :
1028 |   verschoben   INCR i;
1029 |   min schreibpos INCR i;
1030 |   max schreibpos INCR i;
1031 |   cpos         DECR i;
1032 |   output mode := out feld;
1033 |   schreibmarke positionieren (stelle)           (* 11.05.85
+   |   -ws- *)
1034 | END PROC verschiebe;
1035 |

1036 konstantenneuberechnen ... PROC konstanten neu berechnen :
1037 |   min schreibpos := anfang + verschoben;
1038 |   IF min schreibpos < 0                               (* 17.05.85
+   |   -ws- *)
1039 |   THEN min schreibpos DECR verschoben; verschoben := 0
1040 |   FI ;
1041 |   max schreibpos := min schreibpos + laenge - 1 - markierausgleich;
1042 |   cpos := rand + laenge - max schreibpos
1043 | END PROC konstanten neu berechnen;
1044 |

1045 schreibmarkepositionie ... PROC schreibmarke positionieren (INT CONST sstelle) :
1046 |   cursor (cpos + sstelle, zeile)
1047 | END PROC schreibmarke positionieren;
1048 |

1049 simplefeldout ..... PROC simple feldout (TEXT CONST satz, INT CONST dummy) :
1050 |   (* PRECONDITION : NOT markiert AND verschoben = 0 *)
1051 |   (* AND feldrest schon geloescht *)
1052 |   schreibmarke an feldanfang positionieren;
1053 |   out subtext mit randbehandlung (satz, anfang, anfang + laenge - 1);
1054 |   IF (satz SUB LENGTH satz) = blank THEN absatzmarke schreiben
+   |   (TRUE) FI .
1055 |

1056 schreibmarkeanfelfandanfa |schreibmarke an feldanfang positionieren : cursor (rand + 1, zeile) .
1057 | END PROC simple feldout;
1058 |

1059 feldout ..... PROC feldout (TEXT CONST satz, INT CONST sstelle) :
1060 |   schreibmarke an feldanfang positionieren;
1061 |   feld ausgeben;
1062 |   feldrest loeschen;
1063 |   IF (satz SUB LENGTH satz) = blank THEN absatzmarke schreiben
+   |   (TRUE) FI .
1064 |

1065 schreibmarkeanfelfandanfa |schreibmarke an feldanfang positionieren : cursor (rand + 1, zeile) .
1066 |

```

```

1067   feldausgeben      |feld ausgeben :
1068   |                   | INT VAR von := anfang + verschoben, bis := von + laenge - 1;
1069   |                   | IF nicht markiert
1070   |                   | THEN unmarkiert ausgeben
1071   |                   | ELIF markiertes nicht sichtbar
1072   |                   | THEN unmarkiert ausgeben
1073   |                   | ELSE markiert ausgeben
1074   |                   | FI .
1075
1076   nichtmarkiert      |nicht markiert : marke <= 0 .
1077
1078   markiertesnichtsichtba |markiertes nicht sichtbar :
1079   |                   | bis DECR marklength * (1 + SIGN sstelle); marke > bis + 1 .
1080
1081   unmarkiertausgeben   |unmarkiert ausgeben :
1082   |                   | out subtext mit randbehandlung (satz, von, bis) .
1083
1084   markiertausgeben     |markiert ausgeben :
1085   |                   | INT VAR smarke := max (von, marke);
1086   |                   | out text (satz, von, smarke - 1); out (begin mark);
1087   |                   | verschiedene feldout modes behandeln .
1088
1089   verschiedenefeldoutmod |verschiedene feldout modes behandeln :
1090   |                   | IF sstelle = 0
1091   |                   | THEN out subtext mit randbehandlung (satz, smarke, bis); out (end
1092   |                   | mark)
1093   |                   | ELSE out text (satz, smarke, zeilenrand); out (end mark);
1094   |                   | (*sh*)
1095   |                   | out subtext mit randbehandlung (satz, sstelle, bis)
1096   |                   | FI .
1097
1098   zeilenrand           |zeilenrand : min (bis, sstelle - 1) .
1099   |                   | END PROC feldout;
1100
1101   absatzmarkeschreiben ..... |PROC absatzmarke schreiben (BOOL CONST schreiben) :
1102   |                   | IF fliesstext AND nicht markiert
1103   |                   | THEN cursor (rand + 1 + laenge, zeile);
1104   |                   | out (absatzmarke) ;
1105   |                   | absatzmarke steht := TRUE
1106   |                   | FI .
1107
1108   nichtmarkiert      |nicht markiert : marke <= 0 .
1109
1110   absatzmarke         |absatzmarke :
1111   |                   | IF NOT schreiben
1112   |                   | THEN " "
1113   |                   | ELIF marklength > 0
1114   |                   | THEN "15"14"
1115   |                   | ELSE "15" "14" "
1116   |                   | FI .
1117   |                   | END PROC absatzmarke schreiben;

```

```

1117 inversout .....|PROC invers out (TEXT CONST satz, INT CONST von, bis, TEXT CONST
+                |pre, post) :
1118                |IF mark refresh line mode
1119                |THEN feldout (satz, stelle)
1120                |ELSE schreibmarke positionieren (von);
1121                |out (begin mark); markleft; out (pre);
1122                |out text (satz, von, bis - 1); out (post)
1123                |FI .
1124                |
1125                |markleft :
1126                |marklength TIMESOUT left .
1127                |
1128                |END PROC invers out;
1129                |

```

```

1130 feldrestloeschen .....|PROC feldrest loeschen :
1131                |IF rand + laenge < maxbreite COR invertierte darstellung
1132                |THEN INT VAR x; get cursor (x, zeile);
1133                | (rand + laenge - x + 1 + absatz ausgleich) TIMESOUT blank;
+                | (*sh*)
1134                | cursor (x, zeile)
1135                | ELSE out (clear eol); absatzmarke steht := FALSE
1136                | FI
1137                | END PROC feldrest loeschen;
1138                |

```

```

1139 AUFFUELLENMIT .....|OP AUFFUELLENMIT (TEXT VAR satz, TEXT CONST fuellzeichen) :
1140                |INT VAR i;
1141                |FOR i FROM stelle - LENGTH satz DOWNT0 2 REP
1142                |satz CAT fuellzeichen
1143                |PER
1144                |END OP AUFFUELLENMIT;
1145                |

```

```

1146 einrueckposition .....|INT PROC einrueckposition (TEXT CONST satz) :
+                | (*sh*)
1147                |IF fliesstext AND satz = blank
1148                |THEN anfang
1149                |ELSE max (pos (satz, ""33"", ""254"", 1), 1)
1150                |FI
1151                |END PROC einrueckposition;
1152                |

```

```

1153 letzterwortanfang .....|INT PROC letzter wortanfang (TEXT CONST satz) :
+                | (*sh*)
1154                |INT CONST ganz links := max (1, marke);
1155                |BOOL VAR noch nicht im neuen wort := TRUE;
1156                |INT VAR i;
1157                |FOR i FROM stelle DOWNT0 ganz links REP
1158                |IF noch nicht im neuen wort
1159                |THEN noch nicht im neuen wort := char = blank
1160                |ELIF is kanji esc (char)
1161                |THEN LEAVE letzter wortanfang WITH i
1162                |ELIF nicht mehr im neuen wort
1163                |THEN LEAVE letzter wortanfang WITH i + 1
1164                |FI
1165                |PER ;

```

```

1166      | ganz links .
1167      |
1168      char      | char : satz SUB i .
1169      |
1170      nichtmehr | nicht mehr im neuen wort : char = blank COR within kanji (satz, i)
1171      imneuenwort | END PROC letzter wortanfang;
1172      |

```

```

1173      getchar ..... | PROC getchar (TEXT VAR zeichen) :
1174      | IF kommando = ""
1175      | THEN inchar (zeichen); IF lernmodus THEN audit CAT zeichen FI
1176      | ELSE zeichen := kommando SUB kommando zeiger;
1177      | kommando zeiger INCR 1;
1178      | IF kommando zeiger > LENGTH kommando
1179      | THEN kommando zeiger := 1; kommando := ""
1180      | FI ;
1181      | IF LENGTH kommando - kommando zeiger < 3
1182      | THEN kommando CAT inchety
1183      | FI
1184      | FI .
1185      | END PROC getchar;
1186      |

```

```

1187      inchety ..... | TEXT PROC inchety :
1188      | IF lernmodus
1189      | THEN TEXT VAR t := inchar (zeichen); audit CAT t; t
1190      | ELSE inchar (zeichen)
1191      | FI
1192      | END PROC inchety;
1193      |

```

```

1194      isincharety ..... | BOOL PROC is incharety (TEXT CONST muster) :
1195      | IF kommando = ""
1196      | THEN TEXT CONST t := inchar (zeichen);
1197      | IF t = muster THEN TRUE ELSE kommando := t; FALSE FI
1198      | ELIF (kommando SUB kommando zeiger) = muster
1199      | THEN kommando zeiger INCR 1;
1200      | IF kommando zeiger > LENGTH kommando
1201      | THEN kommando zeiger := 1; kommando := ""
1202      | FI ;
1203      | TRUE
1204      | ELSE FALSE
1205      | FI
1206      | END PROC is incharety;
1207      |

```

```

1208      getcharety ..... | TEXT PROC getcharety :
1209      | IF kommando = ""
1210      | THEN inchar (zeichen)
1211      | ELSE TEXT CONST t := kommando SUB kommando zeiger;
1212      | kommando zeiger INCR 1;
1213      | IF kommando zeiger > LENGTH kommando
1214      | THEN kommando zeiger := 1; kommando := ""
1215      | FI ; t
1216      | FI
1217      | END PROC getcharety;

```

1218

```

1219 geteditcursor .....|PROC get editcursor (INT VAR x, y) :
+                       |  (*sh*)
1220                       | IF actual editor > 0 THEN aktualisiere bildparameter FI;
1221                       | x := rand - (anfang + verschoben - 1 - markierausgleich) + stelle;
1222                       | y := zeile .
1223
1224     aktualisierebildparame | aktualisiere bildparameter :
1225                       | INT VAR old x, old y; get cursor (old x, old y);
1226                       | dateizustand holen; bildausgabe steuern; satznr zeigen;
1227                       | fenster zeigen; zeile := bildrand + zeilenr; cursor (old x, old
+                       | y) .
1228                       | END PROC get editcursor;
1229
1230                       | (***** Zugriff auf Feldstatus
+                       | *****).
1231
1232     stelle               | stelle       : feldstatus.stelle .
1233     altestelle           | alte stelle  : feldstatus.alte stelle .
1234     rand                 | rand        : feldstatus.rand .
1235     limit                | limit       : feldstatus.limit .
1236     anfang               | anfang      : feldstatus.anfang .
1237     marke                | marke       : feldstatus.marke .
1238     laenge               | laenge      : feldstatus.laenge .
1239     verschoben           | verschoben  : feldstatus.verschoben .
1240     einfuegen            | einfuegen   : feldstatus.einfuegen .
1241     fliesstext           | fliesstext  : feldstatus.fliesstext .
1242     writeaccess          | write access : feldstatus.write access .
1243     tabulator            | tabulator   : feldstatus.tabulator .
1244
1245                       | (*****
+                       | *****)
1246
1247                       | LET undefinierter bereich = 0,      nix           = 1,
1248                       |     bildzeile           = 2,      akt satznr   = 2,
1249                       |     abschnitt          = 3,      ueberschrift = 3,
1250                       |     bild                = 4,      fehlermeldung = 4;
1251
1252                       | LET BILDSTATUS = STRUCT (INT feldlaenge, kurze feldlaenge,
1253                       |     bildrand, bildlaenge, kurze bildlaenge,
1254                       |     ueberschriftbereich, bildbereich,
1255                       |     erster neusatz, letzter neusatz,
1256                       |     old zeilenr, old lineno, old mark
+                       |     lineno,
1257                       |     BOOL zeileneinfuegen, old line update,
1258                       |     TEXT satznr pre, ueberschrift pre,
1259                       |     ueberschrift text, ueberschrift post,
+                       |     old satz,

```

```

1260                                     FRANGE old range,
1261                                     FILE file),
1262     EDITSTATUS = STRUCT (FELDSTATUS feldstatus, BILDSTATUS
+         bildstatus),
1263     max editor = 10,
1264     EDITSTACK = ROW max editor EDITSTATUS;
1265
1266 BILDSTATUS VAR bildstatus ;
1267 EDITSTACK VAR editstack;
1268
1269 ROW max editor INT VAR einrueckstack;
1270
1271 BOOL VAR markiert;
1272 TEXT VAR filename, tab, bildsatz, bildzeichen, fehlertext,
1273     akt bildsatz ;
1274 INT VAR zeilennr, satznr, bildanfang, bildmarke, feldmarke,
1275     actual editor := 0, max used editor := 0,
1276     letzter editor auf dieser datei,
1277     alte einrueckposition := 1;
1278

1279 aktuellereditor ..... INT PROC aktueller editor : actual editor END PROC aktueller
+
1280     editor;

1281 groesstereeditor ..... INT PROC groesster editor : max used editor END PROC groesster
+
1282     editor;
1283     (***** bildeditor
+         *****)
1284

1285 bildeditor ..... PROC bildeditor (TEXT CONST res, PROC (TEXT CONST) kommando
+         interpreter) :
1286     evtl fehler behandeln;
1287     enable stop;
1288     TEXT VAR reservierte tasten := ""11"12"27"bf" ;
1289     reservierte tasten CAT res ;
1290     INT CONST my highest editor := max used editor;
1291     laenge := feldlaenge;
1292     konstanten neu berechnen;
1293     REP
1294     markierung justieren;
1295     altes feld nachbereiten;
1296     feldlaenge einstellen;
1297     ueberschrift zeigen;
1298     fenster zeigen ;
1299     zeile bereitstellen;
1300     zeile editieren;
1301     kommando ausfuehren
1302     PER .
1303

1304 evtlfehlerbehandeln evtl fehler behandeln :
1305     IF is error
1306     THEN fehlertext := errormessage;
1307     IF fehlertext <> "" THEN neu (fehlermeldung, nix) FI;
1308     clear error
1309     ELSE fehlertext := ""

```

```

1310      |  FI .
1311      |
1312  markierungjustieren | markierung justieren :
1313      |   IF bildmarke > 0
1314      |   THEN IF satznr <= bildmarke
1315      |         THEN bildmarke := satznr;
1316      |           stelle := max (stelle, feldmarke);
1317      |           marke := feldmarke
1318      |         ELSE marke := 1
1319      |         FI
1320      |   FI .
1321      |
1322  zeilebereitstellen  | zeile bereitstellen : IF hinter letztem satz THEN insert record
+      |   (file) FI .
1323  hinterletztensatz  | hinter letztem satz : lineno (file) > lines (file) .
1324      |
1325  altesfeldnachbereiten | altes feld nachbereiten :
1326      |   IF old line update AND lineno (file) <> old lineno
1327      |   THEN IF verschoben <> 0
1328      |         THEN verschoben := 0; konstanten neu berechnen;
1329      |         FI ;
1330      |         INT CONST alte zeilennr := old lineno - bildanfang + 1;
1331      |         IF alte zeilennr > 0 AND alte zeilennr <= aktuelle
+      |           bildlaenge
1332      |         THEN INT CONST m := marke;
1333      |           IF lineno (file) < old lineno
1334      |             THEN marke := 0
1335      |             ELIF old lineno = bildmarke
1336      |             THEN marke := min (feldmarke, LENGTH old_satz + 1)
1337      |             ELSE marke := min (marke, LENGTH old_satz + 1)
1338      |             FI ;
1339      |           zeile := bildrand + alte zeilennr;
1340      |           feldout (old_satz, 0); marke := m
1341      |         FI
1342      |   FI ;
1343      |   old line update := FALSE; old_satz := "" .
1344      |
1345  feldlaengeeinstellen | feldlaenge einstellen :
1346      |   INT CONST alte laenge := laenge;
1347      |   IF zeilennr > kurze bildlaenge
1348      |   THEN laenge := kurze feldlaenge
1349      |   ELSE laenge := feldlaenge
1350      |   FI ;
1351      |   IF laenge <> alte laenge
1352      |   THEN konstanten neu berechnen
1353      |   FI .
1354      |
1355  zeileeditieren      | zeile editieren :
1356      |   zeile := bildrand + zeilennr;
1357      |   exec (PROC (TEXT VAR, TEXT CONST) feldeditor, file, reservierte
+      |     tasten);
1358      |   old lineno := satznr;
1359      |   IF markiert oder verschoben
1360      |   THEN old line update := TRUE; read record (file, old_satz)
1361      |   FI .
1362      |

```

```

1363 markiertoderverschoben | markiert oder verschoben : markiert COR verschoben <> @ .
1364
1365 kommandoausfuehren | kommando ausfuehren :
1366 | getchar (bildzeichen);
1367 | SELECT pos (kommandos, bildzeichen) OF
1368 | CASE x hop : hop kommando verarbeiten
1369 | CASE x esc : esc kommando verarbeiten
1370 | CASE x up : zum vorigen satz
1371 | CASE x down : zum folgenden satz
1372 | CASE x rubin : zeicheneinfuegen umschalten
1373 | CASE x mark : markierung umschalten
1374 | CASE x cr : eingerueckt mit cr (* 08.06.84
+ | -ws- *)
1375 | CASE x inscr : eingerueckt zum folgenden satz
1376 | CASE x abschr : zum anfang des folgenden satzes
1377 | END SELECT .
1378
1379 kommandos | kommandos :
1380 | LET x hop = 1, x up = 2,
1381 | x down = 3, x rubin = 4,
1382 | x cr = 5, x mark = 6,
1383 | x abschr = 7, x inscr = 8,
1384 | x esc = 9;
1385 |
1386 | ""1""3""10""11""13""16""17""18""27"" .
1387
1388 zeicheneinfuegenumscha | zeicheneinfuegen umschalten :
1389 | rubin segment in ueberschrift eintragen;
1390 | neu (ueberschrift, nix) .
1391
1392 rubinsegmentinuebersch | rubin segment in ueberschrift eintragen :
1393 | replace (ueberschrift text, 9, rubin segment) .
1394
1395 rubinsegment | rubin segment :
1396 | IF einfuegen THEN "RUBIN" ELSE "....." FI .
1397
1398 hopkommandoverarbeiten | hop kommando verarbeiten :
1399 | getchar (bildzeichen);
1400 | read record (file, bildsatz);
1401 | SELECT pos (hop kommandos, bildzeichen) OF
1402 | CASE y hop : nach oben
1403 | CASE y cr : neue seite
1404 | CASE y up : zurueckblaettern
1405 | CASE y down : weiterblaettern
1406 | CASE y tab : put tabs (file, tabulator); neu (ueberschrift,
+ | nix)
1407 | CASE y rubout : zeile loeschen
1408 | CASE y rubin : zeileneinfuegen umschalten
1409 | END SELECT .
1410
1411 hopkommandos | hop kommandos :
1412 | LET y hop = 1, y up = 2,
1413 | y tab = 3, y down = 4,
1414 | y rubin = 5, y rubout = 6,
1415 | y cr = 7;

```

```

1416
1417      ""1""3""9""10""11""12""13"" .
1418
1419      zeileneinfuegenumschal  zeileneinfuegen umschalten :
1420                             zeileneinfuegen := NOT zeileneinfuegen;
1421                             IF zeileneinfuegen
1422                             THEN zeile aufspalten; logisches eof setzen
1423                             ELSE leere zeile am ende loeschen; logisches eof loeschen
1424                             FI ; restbild zeigen .
1425
1426      zeileaufspalten         zeile aufspalten :
1427                             IF stelle <= LENGTH bildsatz OR stelle = 1
1428                             THEN loesche ggf trennende blanks und spalte zeile
1429                             FI .
1430
1431      loescheggfrennendebla  loesche ggf trennende blanks und spalte zeile:      (* 26.06.84
+                               -bk- *)
1432      +                         INT VAR first non blank pos := stelle;
1433      +                         WHILE first non blank pos <= length (bildsatz) CAND
1434      +                         (bildsatz SUB first non blank pos) = blank REP
1435      +                         first non blank pos INCR 1
1436      +                         PER ;
1437      +                         split line and indentation;
1438      +                         (*sh*)
1439      +                         first non blank pos := stelle - 1;
1440      +                         WHILE first non blank pos >= 1 CAND
1441      +                         (bildsatz SUB first non blank pos) = blank REP
1442      +                         first non blank pos DECR 1
1443      +                         PER;
1444      +                         bildsatz := subtext (bildsatz, 1, first non blank pos);
1445      +                         write record (file, bildsatz) .
1446
1446      splitlineandindentatio  split line and indentation :
1447      +                         split line (file, first non blank pos, TRUE) .
1448
1449      logischeseofsetzen      logisches eof setzen :
1450      +                         down (file); col (file, 1);
1451      +                         set range (file, 1, 1, old range); up (file) .
1452
1453      leerezeileamendeloesch  leere zeile am ende loeschen :
1454      +                         to line (file, lines (file));
1455      +                         IF len (file) = 0 THEN delete record (file) FI;
1456      +                         to line (file, satznr) .
1457
1458      logischeseofloeschen    logisches eof loeschen :
1459      +                         col (file, stelle); set range (file, old range) .
1460
1461      restbildzeigen          restbild zeigen :
1462      +                         erster neusatz := satznr;
1463      +                         letzter neusatz := bildanfang + bildlaenge - 1;
1464      +                         rest segment in ueberschrift eintragen;
1465      +                         neu (ueberschrift, abschnitt) .
1466

```

```

1467 restsegmentueberschr |rest segment in ueberschrift eintragen :
1468 | replace (ueberschrift text, feldlaenge - 25, rest segment) .
1469 |
1470 restsegment |rest segment :
1471 | IF zeileneinfuegen THEN "REST" ELSE "..." FI .
1472 |
1473 esckommandoverarbeiten |esc kommando verarbeiten :
1474 | getchar (bildzeichen);
1475 | eventuell zeichen zurueckweisen; (* 04.05.85
+ | -ws- *)
1476 | IF taste ist reserviert
1477 | THEN belegte taste ausfuehren
1478 | ELSE fest vordefinierte esc funktion
1479 | FI ; ende nach quit .
1480 |
1481 eventuellzeichenzuruec |eventuell zeichen zurueckweisen : (* 04.05.85
+ | -ws- *)
1482 | IF NOT write access CAND NOT erlaubte taste
1483 | THEN benutzer warnen; LEAVE kommando ausfuehren
1484 | FI .
1485 |
1486 erlaubtetaste |erlaubte taste : pos (zulaessige zeichen, bildzeichen) > 0 .
1487 zulaessigezeichen |zulaessige zeichen : res + "1"2"8"27"bfq" .
1488 benutzerwarnen |benutzer warnen : out (piep) .
1489 |
1490 endenachquit |ende nach quit :
1491 | IF max used editor < my highest editor THEN LEAVE bildeditor FI .
1492 |
1493 tasteistreserviert |taste ist reserviert : pos (res, bildzeichen) > 0 .
1494 |
1495 festvordefinierteescfu |fest vordefinierte esc funktion :
1496 | read record (file, bildsatz);
1497 | SELECT pos (esc kommandos, bildzeichen) OF
1498 | CASE z hop : lernmodus umschalten
1499 | CASE z esc : kommandodialog versuchen
1500 | CASE z left : zum vorigen wort
1501 | CASE z right : zum naechsten wort
1502 | CASE z b : bild an aktuelle zeile angleichen
1503 | CASE z f : belegte taste ausfuehren
1504 | CASE z rubout : markiertes vorsichtig loeschen
1505 | CASE z rubin : vorsichtig geloeschtes einfuegen
1506 | OTHERWISE : belegte taste ausfuehren
1507 | END SELECT .
1508 |
1509 esckommandos |esc kommandos :
1510 | LET z hop = 1, z right = 2,
1511 | z left = 3, z rubin = 4,
1512 | z rubout = 5, z esc = 6,
1513 | z b = 7, z f = 8;
1514 |
1515 | "1"2"8"11"12"27"bf" .
1516 |

```

```

1517 zumvorigenwort |zum vorigen wort :
1518 | IF vorgaenger erlaubt
1519 | THEN vorgaenger; read record (file, bildsatz);
1520 | stelle := LENGTH bildsatz + 1; push (esc + left)
1521 | FI .
1522
1523 vorgaengererlaubt |vorgaenger erlaubt :
1524 | satznr > max (1, bildmarke) .
1525
1526 zumnaechstenwort |zum naechsten wort :
1527 | IF nicht auf letztem satz THEN weitersuchen wenn nicht gefunden FI
+
1528 |
1529 nichtaufletztemsatz |nicht auf letztem satz : line no (file) < lines (file) .
1530
1531 weitersuchenwennnichtig |weitersuchen wenn nicht gefunden :
1532 | nachfolgenden satz holen;
1533 | IF (nachfolgender satz SUB anfang) = blank
1534 | THEN push (abscr + esc + right)
1535 | ELSE push (abscr)
1536 | FI .
1537
1538 nachfolgendensatzholen |nachfolgenden satz holen :
1539 | down (file); read record (file, nachfolgender satz); up (file) .
1540
1541 bildanaktuellezeileang |bild an aktuelle zeile angleichen :
1542 | anfang INCR verschoben; verschoben := 0;
1543 | margin segment in ueberschrift eintragen;
1544 | neu (ueberschrift, bild) .
1545
1546 marginsegmentinuebersc |margin segment in ueberschrift eintragen :
1547 | replace (ueberschrift text, 2, margin segment) .
1548
1549 marginsegment |margin segment :
1550 | IF anfang <= 1
1551 | THEN "....."
1552 | ELSE TEXT VAR margin text := "M" + text (anfang);
1553 | (6 - LENGTH margin text) * "." + margin text
1554 | FI .
1555
1556 belegtetasteausfuehren |belegte taste ausfuehren :
1557 | kommando analysieren (bildzeichen, PROC(TEXT CONST) kommando
+
1558 | interpreter) .
1559
1559 kommandodialogversuche |kommandodialog versuchen:
1560 | IF fenster ist zu schmal fuer dialog
1561 | THEN kommandodialog ablehnen
1562 | ELSE kommandodialog fuehren
1563 | FI .
1564

```

```

1565 fensteristzuschmal fuer fenster ist zu schmal fuer dialog : laenge < 20 .
1566
1567 kommandodialogablehnen kommandodialog ablehnen :
1568 fehler text := "zu schmal fuer ESC ESC"; neu (fehlermeldung, nix)
1569
1570 kommandodialogfuehren kommandodialog fuehren:
1571 INT VAR x0, x1, x2, x3, y;
1572 get cursor (x0, y);
1573 cursor (rand + 1, bildrand + zeilenr);
1574 get cursor (x1, y);
1575 out (begin mark); out (monitor meldung);
1576 get cursor (x2, y);
1577 (laenge - LENGTH monitor meldung - marklength) TIMESOUT blank;
1578 get cursor (x3, y);
1579 out (end mark); out (blank);
1580 kommandozeile editieren;
1581 ueberschrift zeigen;
1582 absatz ausgleich := 2;
+ (*sh*)
1583 IF kommandotext = "" THEN LEAVE kommandodialog fuehren FI;
1584 kommando auf taste legen ("f", kommandotext);
1585 kommando analysieren ("f", PROC(TEXT CONST) kommando interpreter)
1586 IF fehler text <> ""
1587 THEN push (esc + esc + esc + "k")
1588 ELIF markiert
1589 THEN zeile neu
1590 FI .
1591
1592 kommandozeileeditieren kommandozeile editieren :
1593 TEXT VAR kommandotext := "";
1594 cursor (x1, y); out (begin mark);
1595 disable stop;
1596 darstellung invertieren;
1597 editget schleife;
1598 darstellung invertieren;
1599 enable stop;
1600 cursor (x3, y); out (end mark);
1601 exec (PROC (TEXT CONST, INT CONST) feldout, file, stelle);
1602 cursor (x0, y) .
1603
1604 darstellunginvertieren darstellung invertieren :
1605 TEXT VAR dummy := begin mark; begin mark := end mark; end mark :=
+ dummy;
1606 invertierte darstellung := NOT invertierte darstellung .
1607
1608 editgetschleife editget schleife :
1609 TEXT VAR exit char;
1610 REP
1611 cursor (x2, y);
1612 editget (kommandotext, max textlength, rand + laenge - x cursor
1613 "", "k?", exit char);
1614 neu (ueberschrift, nix);
1615 IF exit char = ""27"k"
1616 THEN kommando text := kommando auf taste ("f")
1617 ELIF exit char = ""27"?
1618 THEN TEXT VAR taste; getchar (taste);
1619 kommando text := kommando auf taste (taste)

```

```

1620 |         ELIF exit char = ""27!"
1621 |         THEN getchar (taste);
1622 |             IF ist reservierte taste
1623 |                 THEN set busy indicator;
1624 |                 +
1625 |                 (*sh*)
1626 |                 out ("FEHLER: "" + taste + "" ist reserviert""7""")
1627 |             ELSE kommando auf taste legen (taste, kommandotext);
1628 |                 kommandotext := ""; LEAVE editget schleife
1629 |             FI
1630 |         ELSE LEAVE editget schleife
1631 |         FI
1632 |     PER .

1632 | istreserviertetaste | ist reservierte taste : pos (res, taste) > 0 .

1633 | monitormeldung     | monitor meldung       : "gib kommando : " .
1634 |

1635 | neuseite           | neue seite : bildanfang := satznr; zeilennr := 1; neu (akt satznr,
1636 | +
1637 | bild) .

1637 | weiterblaettern   | weiterblaettern :
1638 | INT CONST akt bildlaenge := aktuelle bildlaenge;
1639 | IF nicht auf letztem satz
1640 | THEN erster neusatz := satznr;
1641 |     IF zeilennr >= akt bildlaenge
1642 |         THEN bildanfang INCR akt bildlaenge; neu (akt satznr, bild)
1643 |     FI ;
1644 |     satznr := min (lines (file), bildanfang + akt bildlaenge - 1);
1645 |     letzter neusatz := satznr;
1646 |     toline (file, satznr);
1647 |     stelle DECR verschoben;
1648 |     neu (akt satznr, nix);
1649 |     zeilennr := satznr - bildanfang + 1;
1650 |     IF markiert THEN neu (nix, abschnitt) FI;
1651 |     einrueckposition bestimmen
1652 |     FI .
1653 |

1654 | zurueckblaettern  | zurueckblaettern :
1655 | IF vogaenger erlaubt
1656 | THEN IF zeilennr <= 1
1657 |     THEN bildanfang := max (1, bildanfang - aktuelle bildlaenge);
1658 |     neu (akt satznr, bild)
1659 |     FI ;
1660 |     nach oben; einrueckposition bestimmen
1661 |     FI .
1662 |

1663 | zeileloeschen     | zeile loeschen :
1664 | IF stelle = 1
1665 | THEN delete record (file);
1666 |     erster neusatz := satznr;
1667 |     letzter neusatz := bildanfang + bildlaenge - 1;
1668 |     neu (nix, abschnitt)
1669 | ELSE zeilen rekombinieren
1670 |     FI .
1671 |

```

```

1672 zeilenrekombinieren | zeilen rekombinieren :
1673 | IF nicht auf letztem satz
1674 | THEN aktuellen satz mit blanks auffuellen;
1675 | delete record (file);
1676 | nachfolgenden satz lesen;
1677 | bildsatz CAT nachfolgender satz ohne fuehrende blanks;
1678 | write record (file, bildsatz);
1679 | erster neusatz := satznr;
1680 | letzter neusatz := bildanfang + bildlaenge - 1;
1681 | neu (nix, abschnitt)
1682 | FI .
1683
1684 aktuellensatzmitblanks | aktuellen satz mit blanks auffuellen :
1685 | bildsatz AUFFUELLENMIT blank .
1686
1687 nachfolgendensatzlesen | nachfolgenden satz lesen :
1688 | TEXT VAR nachfolgender satz;
1689 | read record (file, nachfolgender satz) .
1690
1691 nachfolgendersatzohnef | nachfolgender satz ohne fuehrende blanks :
1692 | satzrest := subtext (nachfolgender satz,
1693 | einrueckposition (nachfolgender satz)); satzrest .
1694
1695 zeileaufsplitten | zeile aufsplitten :
1696 | nachfolgender satz := "";
1697 | INT VAR i;
1698 | FOR i FROM 2 UPTO min (stelle, einrueckposition (bildsatz)) REP
1699 | nachfolgender satz CAT blank
1700 | PER;
1701 | satzrest := subtext (bildsatz, naechste non blank position);
1702 | nachfolgender satz CAT satzrest;
1703 | bildsatz := subtext (bildsatz, 1, stelle - 1);
1704 | write record (file, bildsatz);
1705 | down (file); insert record (file);
1706 | write record (file, nachfolgender satz); up (file) .
1707
1708 naechstenonblankpositi | naechste non blank position :
1709 | INT VAR non blank pos := stelle;
1710 | WHILE (bildsatz SUB non blank pos) = blank REP
1711 | non blank pos INCR 1
1712 | PER; non blank pos .
1713
1714 zumvorigensatz | zum vorigen satz :
1715 | IF vorgaenger erlaubt THEN vorgaenger; einrueckposition bestimmen
1716 | FI .
1717
1717 zumfolgendensatz | zum folgenden satz : (* 12.09.85
1718 | -ws- *)
1718 | IF nachfolger erlaubt THEN nachfolger; einrueckposition bestimmen
1719 | ELSE col (file, len (file) + 1); neu (nix,
1720 | nix)
1721 | FI .

```

```

1722 einrueckpositionbestim einrueckposition bestimmen : (* 27.08.85
+ -ws- *)
1723 read record (file, akt bildsatz);
1724 INT VAR neue einrueckposition := einrueckposition (akt bildsatz);
1725 IF akt bildsatz ist leerzeile
1726 THEN alte einrueckposition := max (stelle, neue einrueckposition)
1727 ELSE alte einrueckposition := min (stelle, neue einrueckposition)
1728 FI .
1729

1730 aktbildsatzistleerzeil akt bildsatz ist leerzeile :
1731 akt bildsatz = "" OR akt bildsatz = blank .
1732

1733 zumanfangdesfolgendens zum anfang des folgenden satzes :
1734 IF nachfolger erlaubt THEN nachfolger; stelle := anfang FI .
1735

1736 nachfolgererlaubt nachfolger erlaubt :
1737 write access COR nicht auf letztem satz .
1738

1739 eingeruecktmitch eingerueckt mit cr :
1740 IF NOT nachfolger erlaubt THEN LEAVE eingerueckt mit cr FI;
+ (*sh*)
1741 read record (file, bildsatz);
1742 INT VAR epos := einrueckposition (bildsatz);
1743 nachfolger; col (file, 1);
1744 IF eof (file)
1745 THEN IF LENGTH bildsatz <= epos
1746 THEN stelle := alte einrueckposition
1747 ELSE stelle := epos
1748 FI
1749 ELSE read record (file, bildsatz);
1750 stelle := einrueckposition (bildsatz);
1751 IF bildsatz ist leerzeile (* 29.08.85
+ -ws- *)
1752 THEN stelle := alte einrueckposition;
1753 aktuellen satz mit blanks auffuellen
1754 FI
1755 FI ;
1756 alte einrueckposition := stelle .
1757

1758 bildsatzistleerzeile bildsatz ist leerzeile :
1759 bildsatz = "" OR bildsatz = blank .
1760

1761 eingeruecktzumfolgende eingerueckt zum folgenden satz :
+ (*sh*)
1762 IF NOT nachfolger erlaubt OR NOT write access
1763 THEN LEAVE eingerueckt zum folgenden satz
1764 FI;
1765 alte einrueckposition merken;
1766 naechsten satz holen;
1767 neue einrueckposition bestimmen;
1768 alte einrueckposition := stelle .
1769

1770 alteinrueckpositionme alte einrueckposition merken :
1771 read record (file, bildsatz);
1772 epos := einrueckposition (bildsatz);

```

```

1773      | auf aufzaehlung pruefen;
1774      | IF epos > LENGTH bildsatz THEN epos := anfang FI.
1775
1776  aufaufzaehlungpruefen  auf aufzaehlung pruefen :
1777      |   BOOL CONST aufzaehlung gefunden :=
1778      |     ist aufzaehlung CAND vorher absatzzeile CAND wort folgt;
1779      |   IF aufzaehlung gefunden THEN epos := anfang des naechsten wortes
1780      |     FI .
1781
1781  istaufzaehlung         |ist aufzaehlung :
1782      | INT CONST wortende := pos (bildsatz, blank, epos, epos + 20) - 1;
1783      | SELECT pos ("*") .", bildsatz SUB wortende) OF
1784      |   CASE 1,2 : wortende = epos
1785      |   CASE 3,4 : wortende <= epos + 7
1786      |   CASE 5 : TRUE
1787      |   OTHERWISE: FALSE
1788      | ENDSSELECT .
1789
1790  vorherabsatzzeile     |vorder absatzzeile :
1791      | IF satznr = 1
1792      | THEN TRUE
1793      | ELSE up (file);
1794      | INT CONST vorige satzlaenge := len (file);
1795      | BOOL CONST vorher war absatzzeile :=
1796      |   subtext (file, vorige satzlaenge, vorige satzlaenge) = blank;
1797      | down (file); vorher war absatzzeile
1798      | FI .
1799
1800  wortfolgt             |wort folgt :
1801      | INT CONST anfang des naechsten wortes :=
1802      |   pos (bildsatz, ""33"", ""254"", wortende + 1);
1803      | anfang des naechsten wortes > wortende .
1804
1805  naechstensatzholen   |naechsten satz holen :
1806      | nachfolger; col (file, 1);
1807      | IF eof (file)
1808      | THEN bildsatz := ""
1809      | ELSE IF neue zeile einfuegen erforderlich
1810      | THEN insert record (file); bildsatz := "";
1811      |   letzter neusatz := bildanfang + bildlaenge - 1
1812      | ELSE read record (file, bildsatz);
1813      |   letzter neusatz := satznr;
1814      |   ggf trennungen zurueckwandeln und umbruch indikator
1815      |   einfuegen
1816      | FI ;
1817      | erster neusatz := satznr;
1818      | neu (nix, abschnitt)
1819      | FI .
1820
1820  neuezeileeinfuegenerfo  neue zeile einfuegen erforderlich :
1821      | BOOL CONST war absatz := war absatzzeile;
1822      | war absatz COR neuer satz ist zu lang .
1823

```

```

1824 warabsatzzeile | war absatzzeile :
1825 | INT VAR wl := pos (kommando, up backcr, kommando zeiger);
1826 | wl = 0 COR (kommando SUB (wl - 1)) = blank .
1827 |
-----
1828 neuersatzistzulang | neuer satz ist zu lang : laenge des neuen satzes >= limit .
1829 |
-----
1830 laengedesneuensatzes | laenge des neuen satzes :
1831 | IF len (file) > 0
1832 | THEN len (file) + wl
1833 | ELSE wl + epos
1834 | FI .
1835 |
-----
1836 upbackcr | up backcr : ""3""20"" .
1837 |
-----
1838 ggfrennungenzurueckwa | ggf trennungen zurueckwandeln und umbruch indikator einfuegen :
1839 | LET trenn k = ""220"";
1840 | trenn strich = ""221"";
1841 | TEXT VAR umbruch indikator;
1842 | IF letztes zeichen ist trenn strich
1843 | THEN entferne trenn strich;
1844 | IF letztes zeichen = trenn k
1845 | THEN wandle trenn k um
1846 | FI ;
1847 | umbruch indikator := up backcr
1848 | ELIF letztes umgebrochenes zeichen ist kanji
1849 | THEN umbruch indikator := up backcr
1850 | ELSE umbruch indikator := blank + up backcr
1851 | FI ;
1852 | change (kommando, wl, wl+1, umbruch indikator) .
1853 |
-----
1854 letztesumgebrochenesze | letztes umgebrochenes zeichen ist kanji : within kanji (kommando,
+ | wl-1) .
1855 |
-----
1856 letzteszeichenisttrenn | letztes zeichen ist trenn strich :
1857 | TEXT CONST last char := letztes zeichen;
1858 | last char = trenn strich COR
1859 | last char = "-" CAND wl > 2 CAND (kommando SUB (wl-2)) <> blank .
1860 |
-----
1861 letzteszeichen | letztes zeichen : kommando SUB (wl-1) .
-----
1862 entfernetrennstrich | entferne trenn strich : delete char (kommando, wl-1); wl DECR 1 .
-----
1863 wandletrennkum | wandle trenn k um : replace (kommando, wl-1, "c") .
-----
1864 loescheindikator | loesche indikator : delete char (kommando, wl) .
1865 |
-----
1866 neueinrueckpositionbe | neue einrueckposition bestimmen :
1867 | IF aufzaehlung gefunden CAND bildsatz ist leerzeile
1868 | THEN stelle := epos
1869 | ELIF NOT bildsatz ist leerzeile
1870 | THEN stelle := einrueckposition (bildsatz)
1871 | ELIF war absatz COR auf letztem satz
1872 | THEN stelle := epos
1873 | ELSE down (file); read record (file, nachfolgender satz);

```

```

1874          up (file); stelle := einrueckposition (nachfolgender satz)
1875          FI ;
1876          IF ist einfuegender aber nicht induzierter umbruch
1877          THEN loesche indikator;
1878          umbruchstelle := stelle + wl - kommando zeiger - anzahl der
+          stz;
1879          umbruchverschoben := 0
1880          FI .
1881
1882 aufletztensatz auf letztem satz : NOT nicht auf letztem satz .
1883
1884 isteinfuegenderabernic ist einfuegender aber nicht induzierter umbruch :
1885 wl := pos (kommando, backcr, kommando zeiger);
1886 wl > 0 CAND (kommando SUB (wl - 1)) <> up char .
1887
1888 anzahlderstz anzahl der stz :
1889 TEXT CONST umgebrochener anfang := subtext (kommando, kommando
+ zeiger, wl-1);
1890 INT VAR anz := 0, anf := pos (umgebrochener anfang, ""1"", ""31"",
+ 1);
1891 WHILE anf > 0 REP
1892   anz INCR 1; anf := pos (umgebrochener anfang, ""1"", ""31"", anf
+   + 1)
1893 PER; anz .
1894
1895 markiertesvorsichtiglo markiertes vorsichtig loeschen :
1896 IF write access CAND markiert
1897 THEN clear removed (file);
1898 IF nur im satz markiert
1899 THEN behandle einen satz
1900 ELSE behandle mehrere saetze
1901 FI
1902 FI .
1903
1904 nurimsatzmarkiert nur im satz markiert : line no (file) = bildmarke .
1905
1906 behandleeinensatz behandle einen satz :
1907 insert record (file);
1908 satzrest := subtext (bildsatz, marke, stelle - 1);
1909 write record (file, satzrest);
1910 remove (file, 1);
1911 change (bildsatz, marke, stelle - 1, "");
1912 stelle := marke;
1913 marke := 0; bildmarke := 0; feldmarke := 0;
1914 markiert := FALSE; mark (file, 0, 0);
1915 konstanten neu berechnen;
1916 IF bildsatz = ""
1917 THEN delete record (file);
1918 erster neusatz := satznr;
1919 letzter neusatz := bildanfang + bildlaenge - 1;
1920 neu (nix, abschnitt)
1921 ELSE write record (file, bildsatz);
1922 neu (nix, bildzeile)
1923 FI .
1924

```

```

1925 behandlemehreresaetze | behandle mehrere saetze :
1926 | erster neusatz := bildmarke;
1927 | letzter neusatz := bildanfang + bildlaenge - 1;
1928 | zeile an aktueller stelle auftrennen;
1929 | ersten markierten satz an markieranfang aufspalten;
1930 | markierten bereich entfernen;
1931 | bild anpassen .
1932 |
1933 zeileanaktuellerstelle | zeile an aktueller stelle auftrennen :
1934 | INT VAR markierte saetze := line no (file) - bildmarke + 1;
1935 | IF nicht am ende der zeile
1936 | THEN IF nicht am anfang der zeile
1937 | THEN zeile aufsplitten
1938 | ELSE up (file); markierte saetze DECR 1
1939 | FI
1940 | FI .
1941 |
1942 nichtamanfangerderzeile | nicht am anfang der zeile : stelle > 1 .
1943 nichtamendederzeile | nicht am ende der zeile : stelle <= LENGTH bildsatz .
1944 |
1945 erstenmarkiertensatzan | ersten markierten satz an markieranfang aufspalten :
1946 | to line (file, line no (file) - (markierte saetze - 1));
1947 | read record (file, bildsatz);
1948 | stelle := feldmarke;
1949 | IF nicht am anfang der zeile
1950 | THEN IF nicht am ende der zeile
1951 | THEN zeile aufsplitten
1952 | ELSE markierte saetze DECR 1
1953 | FI ;
1954 | to line (file, line no (file) + markierte saetze)
1955 | ELSE to line (file, line no (file) + markierte saetze - 1)
1956 | FI ;
1957 | read record (file, bildsatz) .
1958 |
1959 markiertenbereichentfe | markierten bereich entfernen :
1960 | zeilen nr := line no (file) - markierte saetze - bildanfang + 2;
1961 | remove (file, markierte saetze);
1962 | marke := 0; bildmarke := 0; feldmarke := 0;
1963 | markiert := FALSE; mark (file, 0, 0);
1964 | konstanten neu berechnen;
1965 | stelle := 1 .
1966 |
1967 bildanpassen | bild anpassen :
1968 | satz nr := line no (file);
1969 | IF zeilen nr <= 1
1970 | THEN bildanfang := line no (file); zeilen nr := 1;
1971 | neu (akt satznr, bild)
1972 | ELSE neu (akt satznr, abschnitt)
1973 | FI .
1974 |
1975 vorsichtiggeloeschtese | vorsichtig geloeschtes einfuegen :
1976 | IF NOT write access OR removed lines (file) = 0
1977 | THEN LEAVE vorsichtig geloeschtes einfuegen
1978 | FI ;
1979 | IF nur ein satz

```

```

1980      THEN in aktuellen satz einfuegen
1981      ELSE aktuellen satz aufbrechen und einfuegen
1982      FI .
1983
1984  nureinsatz      | nur ein satz : removed lines (file) = 1 .
1985
1986  inaktuellensatz | in aktuellen satz einfuegen :
1987      reinsert (file);
1988      read record (file, nachfolgender satz);
1989      delete record (file);
1990      TEXT VAR t := bildsatz;
1991      bildsatz := subtext (t, 1, stelle - 1);
1992      aktuellen satz mit blanks auffuellen;
1993      +
1994      (*sh*)
1995      bildsatz CAT nachfolgender satz;
1996      satzrest := subtext (t, stelle);
1997      bildsatz CAT satzrest;
1998      write record (file, bildsatz);
1999      stelle INCR LENGTH nachfolgender satz;
2000      neu (nix, bildzeile) .
2001
2002  aktuellensatz | aktuellen satz aufbrechen und einfuegen :
2003      INT CONST alter bildanfang := bildanfang;
2004      old lineno := satznr;
2005      IF stelle = 1
2006      THEN reinsert (file);
2007      read record (file, bildsatz)
2008      ELIF stelle > LENGTH bildsatz
2009      THEN down (file);
2010      reinsert (file);
2011      read record (file, bildsatz)
2012      ELSE INT VAR von := stelle;
2013      WHILE (bildsatz SUB von) = blank REP von INCR 1 PER;
2014      satzrest := subtext (bildsatz, von, LENGTH bildsatz);
2015      INT VAR bis := stelle - 1;
2016      WHILE (bildsatz SUB bis) = blank REP bis DECR 1 PER;
2017      bildsatz := subtext (bildsatz, 1, bis);
2018      write record (file, bildsatz);
2019      down (file);
2020      reinsert (file);
2021      read record (file, bildsatz);
2022      nachfolgender satz := einrueckposition (bildsatz) * blank;
2023      nachfolgender satz CAT satzrest;
2024      down (file); insert record (file);
2025      write record (file, nachfolgender satz); up (file)
2026      FI ;
2027      stelle := max (1, LENGTH bildsatz);
2028      +
2029      (* 22.06.84
2030      -bk- *)
2031      satz nr := line no (file);
2032      zeilennr INCR satznr - old lineno;
2033      zeilennr := min (zeilennr, aktuelle bildlaenge);
2034      bildanfang := satznr - zeilennr + 1;
2035      IF bildanfang veraendert
2036      THEN abschnitt neu (bildanfang, 9999)
2037      ELSE abschnitt neu (old lineno, 9999)
2038      FI ;
2039      neu (akt satznr, nix).

```

```

2036 bildanfangveraendert | bildanfang veraendert : bildanfang <> alter bildanfang .
2037
2038 lernmodusumschalten | lernmodus umschalten :
2039 | learn segment in ueberschrift eintragen; neu (ueberschrift, nix) .
2040
2041 learnsegmentinuebersch | learn segment in ueberschrift eintragen :
2042 | replace (ueberschrift text, feldlaenge - 19, learn segment) .
2043
2044 learnsegment | learn segment :
2045 | IF lernmodus THEN "LEARN" ELSE "....." FI .
2046
2047 markierungumschalten | markierung umschalten :
2048 | IF markiert THEN markierung ausschalten ELSE markierung
+ | einschalten FI .
2049
2050 markierungeinschalten | markierung einschalten :
2051 | bildmarke := satznr; feldmarke := marke; markiert := TRUE;
2052 | mark (file, bildmarke, feldmarke);
2053 | neu (nix, bildzeile) .
2054
2055 markierungausschalten | markierung ausschalten :
2056 | erster neusatz := max (bildmarke, bildanfang);
2057 | letzter neusatz := satznr;
2058 | bildmarke := 0; feldmarke := 0; markiert := FALSE;
2059 | mark (file, 0, 0);
2060 | IF erster neusatz = letzter neusatz
2061 | THEN neu (nix, bildzeile)
2062 | ELSE neu (nix, abschnitt)
2063 | FI .
2064 | END PROC bildeditor;
2065
2066 neu ..... | PROC neu (INT CONST ue bereich, b bereich) :
2067 | ueberschriftbereich := max (ueberschriftbereich, ue bereich);
2068 | bildbereich := max (bildbereich, b bereich)
2069 | END PROC neu;
2070
2071
2072 nachoben ..... | PROC nach oben :
2073 | letzter neusatz := satznr;
2074 | satznr := max (bildanfang, bildmarke);
2075 | toline (file, satznr);
2076 | stelle DECR verschoben;
2077 | zeilenr := satznr - bildanfang + 1;
2078 | erster neusatz := satznr;
2079 | IF markiert
2080 | THEN neu (akt satznr, abschnitt)
2081 | ELSE neu (akt satznr, nix)
2082 | FI
2083 | END PROC nach oben;
2084

```

```

2085  aktuellebildlaenge .....|INT PROC aktuelle bildlaenge :
2086                          |  IF stelle - stelle am anfang < kurze feldlaenge
2087                          |  AND feldlaenge > 0
2088                          |  THEN bildlaenge
+                               |  (*wk*)
2089                          |  ELSE kurze bildlaenge
2090                          |  FI
2091                          |END PROC aktuelle bildlaenge;
2092                          |

2093  vorgaenger .....|PROC vorgaenger :
2094                          |  up (file); satznr DECR 1;
2095                          |  marke := 0; stelle DECR verschoben;
2096                          |  IF zeilenr = 1
2097                          |  THEN bildanfang DECR 1; neu (ueberschrift, bild)
2098                          |  ELSE zeilenr DECR 1; neu (akt satznr, nix);
+                               |  (*sh*)
2099                          |  IF markiert THEN neu (nix, bildzeile) FI
2100                          |  FI
2101                          |END PROC vorgaenger;
2102                          |

2103  nachfolger .....|PROC nachfolger :
2104                          |  down (file); satznr INCR 1;
2105                          |  stelle DECR verschoben;
2106                          |  IF zeilenr = aktuelle bildlaenge
2107                          |  THEN bildanfang INCR 1;
2108                          |  IF rollup erlaubt
2109                          |  THEN rollup
2110                          |  ELSE neu (ueberschrift, bild)
2111                          |  FI
2112                          |  ELSE neu (akt satznr, nix); zeilenr INCR 1
+                               |  (*sh*)
2113                          |  FI ;
2114                          |  IF markiert THEN neu (nix, bildzeile) FI .
2115                          |

2116  rolluperlaubt .....|rollup erlaubt :
2117                          |  kurze bildlaenge = maxlaenge AND kurze feldlaenge = maxbreite .
2118                          |

2119  rollup .....|rollup :
2120                          |  out (down char);
2121                          |  IF bildzeichen = inscr
2122                          |  THEN neu (ueberschrift, nix)
2123                          |  ELIF is cr or down CAND (write access COR nicht auf letztem satz)
+                               |  (*sh*)
2124                          |  THEN neu (nix, bildzeile)
2125                          |  ELSE neu (ueberschrift, bildzeile)
2126                          |  FI .
2127                          |

2128  iscrordown .....|is cr or down :
2129                          |  IF kommando = "" THEN kommando := inchety FI;
2130                          |  kommando char = down char COR kommando char = cr .
2131                          |

2132  kommandochar .....|kommando char : kommando SUB kommando zeiger .
2133                          |

```

```

2134      nichtaufletztensatz      | nicht auf letztem satz : line no (file) < lines (file) .
2135      | END PROC nachfolger;
2136      |
2137      nextincharetyis ..... | BOOL PROC next incharety is (TEXT CONST muster) :
2138      |     INT CONST klen := LENGTH kommando - kommando zeiger + 1,
2139      |           mlen := LENGTH muster;
2140      |     INT VAR i; FOR i FROM 1 UPTO mlen - klen REP kommando CAT inchety
2141      |     PER;
2142      |     subtext (kommando, kommando zeiger, kommando zeiger + mlen - 1) =
2143      |     muster
2144      | END PROC next incharety is;
2145      |
2146      quitlast ..... | PROC quit last: (* 22.06.84
2147      |     -bk- *)
2148      |     IF actual editor > 0 AND actual editor < max used editor
2149      |     THEN verlasse alle groesseren editoren
2150      |     FI .
2151      |
2152      verlasseallegroesseren | verlasse alle groesseren editoren :
2153      |     open editor (actual editor + 1); quit .
2154      | END PROC quit last;
2155      |
2156      quit ..... | PROC quit :
2157      |     IF actual editor > 0 THEN verlasse aktuellen editor FI .
2158      |
2159      verlasseaktuellenedito | verlasse aktuellen editor :
2160      |     disable stop;
2161      |     INT CONST aktueller editor := actual editor;
2162      |     in innersten editor gehen;
2163      |     REP
2164      |     IF zeileneinfuegen THEN hop rubin simulieren FI;
2165      |     ggf bildschirmdarstellung korrigieren;
2166      |     innersten editor schliessen
2167      |     UNTIL aktueller editor > max used editor PER;
2168      |     actual editor := max used editor .
2169      |
2170      ininnersteneditorgehen | in innersten editor gehen : open editor (max used editor) .
2171      |
2172      hoprubinsimulieren | hop rubin simulieren :
2173      |     zeileneinfuegen := FALSE;
2174      |     leere zeilen am dateiende loeschen;
2175      |     (*sh*)
2176      |     ggf bildschirmdarstellung korrigieren;
2177      |     logisches eof loeschen .
2178      |
2179      innersteneditorschlies | innersten editor schliessen :
2180      |     max used editor DECR 1;
2181      |     IF max used editor > 0
2182      |     THEN open editor (max used editor);
2183      |     bildeinschraenkung aufheben
2184      |     FI .

```

```

2181
2182   logischeseofloeschen | logisches eof loeschen :
2183   | col (file, stelle); set range (file, old range) .
2184
2185   leerezeilenamdateiende | leere zeilen am dateiende loeschen : (* 15.08.85
+   | -ws- *)
2186   | satz nr := line no (file) ;
2187   | to line (file, lines (file)) ;
2188   | WHILE lines (file) > 1 AND bildsatz ist leerzeile REP
2189   | delete record (file);
2190   | to line (file, lines (file))
2191   | PER;
2192   | toline (file, satznr) .
2193
2194   bildsatzistleerzeile | bildsatz ist leerzeile :
2195   | TEXT VAR bildsatz;
2196   | read record (file, bildsatz);
2197   | ist leerzeile .
2198
2199   istleerzeile | ist leerzeile :
2200   | bildsatz = "" OR bildsatz = blank .
2201
2202   ggfbildschirmdarstellu | ggf bildschirmdarstellung korrigieren :
2203   | satz nr DECR 1; (* für
+   | Bildschirmkorrektur *)
2204   | IF satznr > lines (file)
2205   | THEN zeilen nr DECR satz nr - lines (file);
2206   | satz nr := lines (file);
2207   | dateizustand retten
2208   | FI .
2209
2210   bildeinschraenkungaufh | bildeinschraenkung aufheben :
2211   | laenge := feldlaenge;
2212   | kurze feldlaenge := feldlaenge;
2213   | kurze bildlaenge := bildlaenge;
2214   | neu (nix, bild) .
2215   | END PROC quit;
2216
2217   nichtsneu ..... |PROC nichts neu : neu (nix, nix) END PROC nichts neu
2218
2219   satznrneu ..... |PROC satznr neu : neu (akt satznr, nix) END PROC satznr neu
2220
2221   ueberschriftneu ..... |PROC ueberschrift neu : neu (ueberschrift, nix) END PROC
+   | ueberschrift neu;
2222
2223   zeileneu ..... |PROC zeile neu :
2224   | INT CONST zeile := line no (file);
2225   | abschnitt neu (zeile, zeile)

```

2226 |END PROC zeile neu;
2227 |

2228 abschnittneu|PROC abschnitt neu (INT CONST von satznr, bis satznr) :
2229 | IF von satznr <= bis satznr
2230 | THEN erster neusatz := min (erster neusatz, von satznr);
2231 | letzter neusatz := max (letzter neusatz, bis satznr);
2232 | neu (nix, abschnitt)
2233 | ELSE abschnitt neu (bis satznr, von satznr)
2234 | FI
2235 |END PROC abschnitt neu;
2236 |

2237 bildabschnittneu|PROC bildabschnitt neu (INT CONST von zeile, bis zeile) :
+ | (*sh*)
2238 | IF von zeile <= bis zeile
2239 | THEN erster neusatz := max (1, von zeile + bildanfang - 1);
2240 | letzter neusatz := min (bildlaenge, bis zeile + bildanfang -
+ | 1);
2241 | IF von zeile < 1
2242 | THEN neu (ueberschrift, abschnitt)
2243 | ELSE neu (nix , abschnitt)
2244 | FI
2245 | ELSE bildabschnitt neu (bis zeile, von zeile)
2246 | FI
2247 |END PROC bildabschnitt neu;
2248 |

2249 bildneu|PROC bild neu : neu (nix, bild) END PROC bild neu;
+ | (*sh*)
2250 |

2251 bildneu|PROC bild neu (FILE VAR f) :
2252 | INT CONST editor no := abs (editinfo (f)) DIV 256;
2253 | IF editor no > 0 AND editor no <= max used editor
2254 | THEN IF editor no = actual editor
2255 | THEN bild neu
2256 | ELSE editstack (editor no).bildstatus.bildbereich := bild
2257 | FI
2258 | FI
2259 |END PROC bild neu;
2260 |

2261 allesneu|PROC alles neu :
2262 | neu (ueberschrift, bild);
2263 | INT VAR i;
2264 | FOR i FROM 1 UPTO max used editor REP
2265 | editstack (i).bildstatus.bildbereich := bild;
2266 | editstack (i).bildstatus.ueberschriftbereich := ueberschrift
2267 | PER
2268 |END PROC alles neu;
2269 |

2270 satznrzeigen|PROC satznr zeigen :
2271 | out (satznr pre); out (text (text (lineno (file)), 4))
2272 |END PROC satznr zeigen;

2273

```

2274 ueberschriftzeigen .....|PROC ueberschrift zeigen :
2275 |   SELECT ueberschriftbereich OF
2276 |     CASE akt satznr      : satznr zeigen;
2277 |       ueberschriftbereich := nix
2278 |     CASE ueberschrift  : ueberschrift schreiben;
2279 |       ueberschriftbereich := nix
2280 |     CASE fehlermeldung : fehlermeldung schreiben;
2281 |       ueberschriftbereich := ueberschrift
2282 |   END SELECT
2283 |END PROC ueberschrift zeigen;
2284
```

```

2285 fensterzeigen .....|PROC fenster zeigen :
2286 |   SELECT bildbereich OF
2287 |     CASE bildzeile :
2288 |       zeile := bildrand + zeilenr;
2289 |       IF line no (file) > lines (file)
2290 |       THEN feldout ("", stelle)
2291 |       ELSE exec (PROC (TEXT CONST, INT CONST) feldout, file,
2292 |         +           stelle)
2293 |       FI
2294 |     CASE abschnitt :
2295 |       bild ausgeben
2296 |     CASE bild :
2297 |       erster neusatz := 1;
2298 |       letzter neusatz := 9999;
2299 |       bild ausgeben
2300 |     OTHERWISE :
2301 |       LEAVE fenster zeigen
2302 |   END SELECT;
2303 |   erster neusatz := 9999;
2304 |   letzter neusatz := 0;
2305 |   bildbereich := nix
2306 |END PROC fenster zeigen ;
```

```

2307 bildausgeben .....|PROC bild ausgeben :
2308 |   BOOL CONST schreiben ist ganz einfach := NOT markiert AND
2309 |     + verschoben = 0;
2310 |   INT CONST save marke := marke,
2311 |     save verschoben := verschoben,
2312 |     save laenge := laenge,
2313 |     act lineno := lineno (file),
2314 |     von := max (1, erster neusatz - bildanfang + 1);
2315 |   INT VAR bis := min (letzter neusatz - bildanfang + 1,
2316 |     + bildlaenge);
2317 |   IF kurze feldlaenge <= 0 THEN bis := min (bis, kurze bildlaenge)
2318 |   FI;
2319 |   IF von > bis THEN LEAVE bild ausgeben FI;
2320 |   verschoben := 0;
2321 |   IF markiert
2322 |   THEN IF mark lineno (file) < bildanfang + von - 1
2323 |     THEN marke := anfang
2324 |     ELSE marke := 0
2325 |     FI
2326 |   FI ;
2327 |   abschnitt loeschen und neuschreiben;
```

2325		to line (file, act lineno);
2326		laenge := save laenge;
2327		verschoben := save verschoben;
2328		marke := save marke .
2329		
2330	markiert	markiert : mark lineno (file) > 0 .
2331		
2332	abschnittloeschenundne	abschnitt loeschen und neuschreiben :
2333		abschnitt loeschen;
2334		INT VAR line number := bildanfang + von - 1;
2335		to line (file, line number);
2336		abschnitt schreiben .
2337		
2338	abschnittloeschen	abschnitt loeschen :
2339		cursor (rand + 1, bildrand + von);
2340		IF bildrest darf komplett geloescht werden
2341		THEN out (clear eop)
2342		ELSE zeilenweise loeschen
2343		FI .
2344		
2345	bildrestdarfkomplettge	bildrest darf komplett geloescht werden :
2346		bis = maxlaenge AND kurze bildlaenge = maxlaenge
2347		AND kurze feldlaenge = maxbreite .
2348		
2349	zeilenweiseloeschen	zeilenweise loeschen :
2350		INT VAR i;
2351		FOR i FROM von UPTO bis REP
2352		check for interrupt;
2353		feldlaenge einstellen;
2354		feldrest loeschen;
2355		IF i < bis THEN out (down char) FI
2356		PER .
2357		
2358	feldlaengeeinstellen	feldlaenge einstellen :
2359		IF ganze zeile sichtbar
2360		THEN laenge := feldlaenge
2361		ELSE laenge := kurze feldlaenge
2362		FI .
2363		
2364	ganzezeilesichtbar	ganze zeile sichtbar : i <= kurze bildlaenge .
2365		
2366	abschnittschreiben	abschnitt schreiben :
2367		INT CONST last line := lines (file);
2368		FOR i FROM von UPTO bis
2369		WHILE line number <= last line REP
2370		check for interrupt;
2371		feldlaenge einstellen;
2372		zeile schreiben;
2373		down (file);
2374		line number INCR 1
2375		PER .
2376		

```

2377 checkforinterrupt | check for interrupt :
2378 | kommando CAT inchety;
2379 | IF kommando < > ""
2380 | THEN IF zeilen nr = 1 CAND up command CAND vorgaenger erlaubt
2381 | THEN LEAVE abschnitt loeschen und neuschreiben
2382 | ELIF zeilen nr = bildlaenge CAND down command CAND nicht
+ | letzter satz
2383 | THEN LEAVE abschnitt loeschen und neuschreiben
2384 | FI
2385 | FI .
2386
2387 vorgaengererlaubt | vorgaenger erlaubt :
2388 | satznr > max (1, bildmarke) .
2389
2390 upcommand | up command : next incharety is ("3") COR next incharety is
+ | ("13") .
2391 | .
2392 downcommand | down command :
2393 | next incharety is ("10") CAND bildlaenge < maxlaenge
2394 | COR next incharety is ("110") .
2395
2396 nichtletzttersatz | nicht letzter satz : act lineno < lines (file) .
2397
2398 zeileschreiben | zeile schreiben :
2399 | zeile := bildrand + i;
2400 | IF schreiben ist ganz einfach
2401 | THEN exec (PROC (TEXT CONST, INT CONST) simple feldout, file, 0)
2402 | ELSE zeile kompliziert schreiben
2403 | FI ;
2404 | IF line number = old lineno THEN old line update := FALSE FI .
2405
2406 zeilekompliziertschrei | zeile kompliziert schreiben :
2407 | IF line number = mark lineno (file) THEN marke := mark col
+ | (file) FI;
2408 | IF line number = act lineno
2409 | THEN verschoben := save verschoben;
2410 | exec (PROC (TEXT CONST, INT CONST) feldout, file, stelle);
2411 | verschoben := 0; marke := 0
2412 | ELSE exec (PROC (TEXT CONST, INT CONST) feldout, file, 0);
2413 | IF line number = mark lineno (file) THEN marke := anfang FI
2414 | FI .
2415 | END PROC bild ausgeben;
2416
2417 bildzeigen ..... | PROC bild zeigen :
+ | (* wk *)
2418 |
2419 | dateizustand holen ;
2420 | ueberschrift zeigen ;
2421 | bildausgabe steuern ;
2422 | bild neu ;
2423 | fenster zeigen ;
2424 | oldline no := satznr ;
2425 | old line update := FALSE ;
2426 | old satz := "" ;

```

```

2427 | old zeilenr := satznr - bildanfang + 1 ;
2428 | dateizustand retten .
2429 |
2430 | ENDPROC bild zeigen ;
2431 |
2432 ueberschriftinitialisi ... | PROC ueberschrift initialisieren :
+ | (*sh*)
2433 | satznr pre :=
2434 | cursor pos + code (bildrand - 1) + code (rand + feldlaenge -
+ | 6);
2435 | ueberschrift pre :=
2436 | cursor pos + code (bildrand - 1) + code (rand) + mark anf;
2437 | ueberschrift text := ""; INT VAR i;
2438 | FOR i FROM 16 UPTO feldlaenge REP ueberschrift text CAT "." PER;
2439 | ueberschrift post := blank + mark end + "Zeile " + mark anf;
2440 | ueberschrift post CAT blank + mark end + " ";
2441 | filename := headline (file);
2442 | filename := subtext (filename, 1, feldlaenge - 24);
2443 | insert char (filename, blank, 1); filename CAT blank;
2444 | replace (ueberschrift text, filenamepos, filename);
2445 | rubin segment in ueberschrift eintragen;
2446 | margin segment in ueberschrift eintragen;
2447 | rest segment in ueberschrift eintragen;
2448 | learn segment in ueberschrift eintragen .
2449 |
2450 filenamepos | filenamepos : (LENGTH ueberschrift text - LENGTH filename + 3)
+ | DIV 2 .
2451 markanf | mark anf : begin mark + mark ausgleich.
2452 markend | mark end : end mark + mark ausgleich.
2453 markausgleich | mark ausgleich : (1 - sign (max (mark size, 0))) * blank .
2454 |
2455 rubinsegmentinuebersch | rubin segment in ueberschrift eintragen :
2456 | replace (ueberschrift text, 9, rubin segment) .
2457 |
2458 rubinsegment | rubin segment :
2459 | IF einfuegen THEN "RUBIN" ELSE "....." FI .
2460 |
2461 marginsegmentinuebersc | margin segment in ueberschrift eintragen :
2462 | replace (ueberschrift text, 2, margin segment) .
2463 |
2464 marginsegment | margin segment :
2465 | IF anfang <= 1
2466 | THEN "....."
2467 | ELSE TEXT VAR margin text := "M" + text (anfang);
2468 | (6 - LENGTH margin text) * "." + margin text
2469 | FI .
2470 |
2471 restsegmentinueberschr | rest segment in ueberschrift eintragen :
2472 | replace (ueberschrift text, feldlaenge - 25, rest segment) .
2473 |

```

```

2474 restsegment |rest segment :
2475 | IF zeileneinfuegen THEN "REST" ELSE "...." FI .
2476 |
2477 learnsegmentinuebersch |learn segment in ueberschrift eintragen :
2478 | replace (ueberschrift text, feldlaenge - 19, learn segment) .
2479 |
2480 learnsegment |learn segment :
2481 | IF lernmodus THEN "LEARN" ELSE "....." FI .
2482 |
2483 |END PROC ueberschrift initialisieren;
2484 |
2485 ueberschriftschreiben ....|PROC ueberschrift schreiben :
2486 | replace (ueberschrift post, satznr pos, text (text (lineno
+ | (file)), 4));
2487 | out (ueberschrift pre); out (ueberschrift text); out (ueberschrift
+ | post);
2488 | get tabs (file, tab);
2489 | IF pos (tab, dach) > 0
2490 | THEN out (ueberschrift pre);
2491 | out subtext (tab, anfang + 1, anfang + feldlaenge - 1);
2492 | cursor (rand + 1 + feldlaenge, bildrand); out (end mark)
2493 | FI .
2494 |
2495 satznrpos | satznr pos : IF mark size > 0 THEN 9 ELSE 10 FI .
+ | (*sh*)
2496 |END PROC ueberschrift schreiben;
2497 |
2498 fehlermeldungschreiben ...|PROC fehlermeldung schreiben :
2499 | ueberschrift schreiben;
2500 | out (ueberschrift pre);
2501 | out ("FEHLER: ");
2502 | out subtext (fehlertext, 1, feldlaenge - 21);
2503 | out (blank);
2504 | out (piep);
2505 | cursor (rand + 1 + feldlaenge, bildrand); out (end mark)
2506 |END PROC fehlermeldung schreiben;
2507 |
2508 setbusyindicator .....|PROC set busy indicator :
2509 | cursor (rand + 2, bildrand)
2510 |END PROC set busy indicator;
2511 |
2512 kommandoanalysieren .....|PROC kommando analysieren (TEXT CONST taste,
2513 | PROC (TEXT CONST) kommando interpreter) :
2514 | disable stop;
2515 | bildausgabe normieren;
2516 | zustand in datei sichern;
2517 | editfile modus setzen;
2518 | kommando interpreter (taste);
2519 | editfile modus zuruecksetzen;
2520 | IF actual editor <= 0 THEN LEAVE kommando analysieren FI;
2521 | absatz ausgleich := 2;
+ | (*sh*)

```

```

2522      | konstanten neu berechnen;
2523      | neues bild bei undefinierter benutzeraktion;
2524      | evtl fehler behandeln;
2525      | zustand aus datei holen;
2526      | bildausgabe steuern .
2527

2528      editfilemodussetzen | editfile modus setzen :
2529      |   BOOL VAR alter editget modus := editget modus ;
2530      |   editget modus := FALSE .
2531

2532      editfilemoduszurueckse | editfile modus zuruecksetzen :
2533      |   editget modus := alter editget modus .
2534

2535      evtlfehlerbehandeln | evtl fehler behandeln :
2536      |   IF is error
2537      |   THEN fehlertext := errormessage;
2538      |     IF fehlertext <> "" THEN neu (fehlermeldung, nix) FI;
2539      |     clear error
2540      |   ELSE fehlertext := ""
2541      |   FI .
2542

2543      zustandindateisichern | zustand in datei sichern :
2544      |   old zeilennr := zeilennr;
2545      |   old mark lineno := bildmarke;
2546      |   dateizustand retten .
2547

2548      zustandausdateiholen | zustand aus datei holen :
2549      |   dateizustand holen;
2550      |   IF letzter editor auf dieser datei <> actual editor
2551      |   THEN zurueck auf alte position; neu (ueberschrift, bild)
2552      |   FI .
2553

2554      zurueckaufalteposition | zurueck auf alte position :
2555      |   to line (file, old lineno);
2556      |   col (file, alte stelle);
2557      |   IF fliesstext
2558      |   THEN editinfo (file, old zeilennr)
2559      |   ELSE editinfo (file, - old zeilennr)
2560      |   FI ; dateizustand holen .
2561

2562      bildausgabenormieren | bildausgabe normieren :
2563      |   bildbereich := undefinierter bereich;
2564      |   erster neusatz := 9999;
2565      |   letzter neusatz := 0 .
2566

2567      neuesbildbeiundefinier | neues bild bei undefinierter benutzeraktion :
2568      |   IF bildbereich = undefinierter bereich THEN alles neu FI .
2569      |   END PROC kommando analysieren;
2570

2571      bildausgabesteuern ..... | PROC bildausgabe steuern :
2572      |   IF markiert
2573      |   THEN IF old mark lineno = 0

```

```

2574 |         THEN abschnitt neu (bildmarke, satznr);
2575 |             konstanten neu berechnen
2576 |         ELIF stelle veraendert
      |             (*sh*)
      |         THEN zeile neu
2577 |             FI
2578 |
2579 |         ELIF old mark lineno > 0
2580 |         THEN abschnitt neu (old mark lineno, (max (satznr, old lineno)));
2581 |             konstanten neu berechnen
2582 |         FI ;
2583 |         IF satznr <> old lineno
2584 |         THEN neu (akt satznr, nix);
2585 |             neuen bildaufbau bestimmen
2586 |         ELSE zeilennr := old zeilennr
2587 |         FI ;
2588 |         zeilennr := min (min (zeilennr, satznr), aktuelle bildlaenge);
2589 |         bildanfang := satznr - zeilennr + 1 .
2590 |
2591 |
2592 |
2593 |
2594 |
2595 |
2596 |
2597 |
2598 |
2599 |
2600 |
2601 |
2602 |
2603 |
2604 |
2605 |
2606 |
2607 |
2608 |
2609 |
2610 |
2611 |
2612 |
2613 |
2614 |
2615 |
2616 |
2617 |
2618 |
2619 |
2620 |
2621 |
2622 |

```

```

2623 wordwrap .....|BOOL PROC word wrap :
+                |    (*sh*)
2624                |    IF actual editor = 0
2625                |    THEN std fliesstext
2626                |    ELSE fliesstext
2627                |    FI
2628                |END PROC word wrap;
2629                |
2630 margin .....|INT PROC margin : anfang END PROC margin;
2631                |
2632 margin .....|PROC margin (INT CONST i) :
+                |    (*sh*)
2633                |    IF anfang <> i CAND i > 0 AND i < 16001
2634                |    THEN anfang := i; neu (ueberschrift, bild);
2635                |    margin segment in ueberschrift eintragen
2636                |    ELSE IF i >= 16001 OR i < 0
2637                |    THEN errorstop ("ungueltige Anfangsposition (1 - 16000)")
2638                |    FI
2639                |    FI .
2640                |
2641 marginsegmentinuebersc|margin segment in ueberschrift eintragen :
2642                |    replace (ueberschrift text, 2, margin segment) .
2643                |
2644 marginsegment        |margin segment :
2645                |    IF anfang <= 1
2646                |    THEN "....."
2647                |    ELSE TEXT VAR margin text := "M" + text (anfang);
2648                |    (6 - LENGTH margin text) * "." + margin text
2649                |    FI .
2650                |
2651                |END PROC margin;
2652                |
2653 rubinmode .....|BOOL PROC rubin mode : rubin mode (actual editor) END PROC rubi'
+                |    mode;
2654                |
2655 rubinmode .....|BOOL PROC rubin mode (INT CONST editor nr) :
+                |    (*sh*)
2656                |    IF editor nr < 1 OR editor nr > max used editor
2657                |    THEN errorstop ("Editor nicht eroeffnet")
2658                |    FI ;
2659                |    IF editor nr = actual editor
2660                |    THEN einfuegen
2661                |    ELSE editstack (editor nr).feldstatus.einfuegen
2662                |    FI
2663                |END PROC rubin mode;
2664                |
2665 edit .....|PROC edit (INT CONST i, TEXT CONST res,
2666                |    PROC (TEXT CONST) kommando interpreter) :
2667                |    edit (i, i, res, PROC (TEXT CONST) kommando interpreter)
2668                |END PROC edit;

```

2669

```

2670 edit .....|PROC edit (INT CONST von, bis, start, TEXT CONST res,
2671              |      PROC (TEXT CONST) kommando interpreter) :
2672              |  disable stop;
2673              |  IF von < bis
2674              |  THEN edit (von+1, bis, start, res, PROC (TEXT CONST) kommando
                |  + interpreter);
2675              |      IF max used editor < von THEN LEAVE edit FI;
2676              |      open editor (von)
2677              |  ELSE open editor (start)
2678              |  FI ;
2679              |  absatz ausgleich := 2;
2680              |  bildeditor (res, PROC (TEXT CONST) kommando interpreter);
2681              |  cursor (1, schirmhoehe);
2682              |  IF is error
2683              |  THEN kommando zeiger := 1; kommando := ""; quit
2684              |  FI ;
2685              |  IF lernmodus CAND actual editor = 0 THEN warnung ausgeben FI .
                |  (*sh*)
2686
2687 warnungausgeben      |  warnung ausgeben :
2688                      |  out (clear eop); out ("WARNUNG: Lernmodus nicht
                |  + ausgeschaltet"13"10") .
2689                      |  END PROC edit;
2690
2691 dateizustandholen .....|PROC dateizustand holen :
2692                      |  modify (file);
2693                      |  get tabs (file, tabulator);
2694                      |  zeilennr und fliesstext und letzter editor aus editinfo decodieren;
2695                      |  limit := max line length (file);
2696                      |  stelle := col (file);
2697                      |  markiert := mark (file);
2698                      |  IF markiert
2699                      |  THEN markierung holen
2700                      |  ELSE keine markierung
2701                      |  FI ;
2702                      |  satz nr := lineno (file);
2703                      |  IF zeilennr > aktuelle bildlaenge
                |  + (*sh*)
2704                      |  THEN zeilennr := min (satznr, aktuelle bildlaenge); bild neu
2705                      |  ELIF zeilennr > satznr
2706                      |  THEN zeilennr := min (satznr, aktuelle bildlaenge)
2707                      |  FI ; zeilennr := max (zeilennr, 1);
2708                      |  bildanfang := satz nr - zeilennr + 1 .
2709
2710 zeilenrundfließstextu |  zeilennr und fliesstext und letzter editor aus editinfo decodieren :
2711                      |  zeilennr := edit info (file);
2712                      |  IF zeilennr = 0
2713                      |  THEN zeilennr := 1;
2714                      |      fliesstext := std fliesstext
2715                      |  ELIF zeilennr > 0
2716                      |  THEN fliesstext := TRUE
2717                      |  ELSE zeilennr := - zeilennr;
2718                      |      fliesstext := FALSE
2719                      |  FI ;
2720                      |  letzter editor auf dieser datei := zeilennr DIV 256;

```

```

2721      | zeilenr := zeilenr MOD 256 .
2722      |
2723      | markierungholen | markierung holen :
2724      | bildmarke := mark lineno (file);
2725      | feldmarke := mark col (file);
2726      | IF line no (file) <= bildmarke
2727      | THEN to line (file, bildmarke);
2728      | marke := feldmarke;
2729      | stelle := max (stelle, feldmarke)
2730      | ELSE marke := 1
2731      | FI .
2732      |
2733      | keinemarkierung | keine markierung :
2734      | bildmarke := 0;
2735      | feldmarke := 0;
2736      | marke := 0 .
2737      | END PROC dateizustand holen;
2738      |
2739      | dateizustandretten ..... | PROC dateizustand retten :
2740      | put tabs (file, tabulator);
2741      | IF fliesstext
2742      | THEN editinfo (file, zeilenr + actual editor * 256)
2743      | ELSE editinfo (file, - (zeilenr + actual editor * 256))
2744      | FI ;
2745      | max line length (file, limit);
2746      | col (file, stelle);
2747      | IF markiert
2748      | THEN mark (file, bildmarke, feldmarke)
2749      | ELSE mark (file, 0, 0)
2750      | FI
2751      | END PROC dateizustand retten;
2752      |
2753      | openeditor ..... | PROC open editor (FILE CONST new file, BOOL CONST access) :
2754      | disable stop; quit last;
2755      | neue bildparameter bestimmen;
2756      | open editor (actual editor + 1, new file, access, x, y, x len, y
2757      | + len).
2758      |
2759      | neubildparameterbesti | neue bildparameter bestimmen :
2760      | INT VAR x, y, x len, y len;
2761      | IF actual editor > 0
2762      | THEN teilbild des aktuellen editors
2763      | ELSE volles bild
2764      | FI .
2765      |
2766      | teilbilddesaktuellened | teilbild des aktuellen editors :
2767      | get editcursor (x, y); bildgroesse bestimmen;
2768      | IF fenster zu schmal
2769      | + (*sh*)
2770      | THEN enable stop; errorstop ("Fenster zu klein")
2771      | ELIF fenster zu kurz
2772      | THEN verkuerztes altes bild nehmen
2773      | FI .

```

```

2773 bildgroessebestimmen bildgroesse bestimmen :
2774     x len := rand + feldlaenge - x + 3;
2775     y len := bildrand + bildlaenge - y + 1 .
2776
2777 fensterzuschmal fenster zu schmal : x > schirmbreite - 17 .
2778 fensterzukurz fenster zu kurz : y > schirmhoehe - 1 .
2779
2780 verkuerztesaltesbildne verkuerztes altes bild nehmen :
2781     x := rand + 1; y := bildrand + 1;
2782     IF fenster zu kurz THEN enable stop; errorstop ("Fenster zu
+ klein") FI;
2783     x len := feldlaenge + 2;
2784     y len := bildlaenge;
2785     kurze feldlaenge := 0;
2786     kurze bildlaenge := 1 .
2787
2788 vollesbild volles bild :
2789     x := 1; y := 1; x len := schirmbreite; y len := schirmhoehe .
2790     END PROC open editor;
2791
2792 openeditor ..... PROC open editor (INT CONST editor nr,
2793     FILE CONST new file, BOOL CONST access,
2794     INT CONST x start, y, x len start, y len) :
2795     INT VAR x := x start,
2796     x len := x len start;
2797     IF editor nr > max editor
2798     THEN errorstop ("zu viele Editor-Fenster")
2799     ELIF editor nr > max used editor + 1 OR editor nr < 1
2800     THEN errorstop ("Editor nicht eroeffnet")
2801     ELIF fenster ungueltig
2802     THEN errorstop ("Fenster ungueltig")
2803     ELSE neuen editor stacken
2804     FI .
2805
2806 fensterungueltig fenster ungueltig :
2807     x < 1 COR x > schirmbreite COR y < 1 COR y > schirmhoehe COR
2808     x len - 2 <= 15 COR y len - 1 < 1 COR
2809     x + x len - 1 > schirmbreite COR y + y len - 1 > schirmhoehe .
2810
2811 neueneditorstacken neuen editor stacken :
2812     disable stop;
2813     IF actual editor > 0 AND ist einschraenkung des alten bildes
2814     THEN dateizustand holen;
2815     aktuelles editorbild einschraenken;
2816     arbeitspunkt in das restbild positionieren;
2817     abgrenzung beruecksichtigen
2818     FI ;
2819     aktuellen zustand retten;
2820     neuen zustand setzen;
2821     neues editorbild zeigen;
2822     actual editor := editor nr;
2823     IF actual editor > max used editor
2824     THEN max used editor := actual editor
2825     FI .

```

```

2826
2827   isteinschraenkungdesal list einschraenkung des alten bildes :
2828       x > rand      CAND  x + x len = rand + feldlaenge + 3  CAND
2829       y > bildrand  CAND  y + y len = bildrand + bildlaenge + 1 .
2830
2831   aktuelleseditorbildein aktuelles editorbild einschraenken :
2832       kurze feldlaenge := x - rand - 3;
2833       kurze bildlaenge := y - bildrand - 1 .
2834
2835   arbeitspunktindasrestb arbeitspunkt in das restbild positionieren :
2836       IF  stelle > 3
2837       THEN stelle DECR 3; alte stelle := stelle
2838       ELSE WHILE zeilennr > 1 AND zeilennr > kurze bildlaenge REP
2839           vorgaenger
2840           PER; old lineno := satznr
2841       FI .
2842
2843   abgrenzungberuecksicht abgrenzung beruecksichtigen :
2844       IF  x - rand > 1
2845       THEN stelle malen;
2846           x INCR 2;
2847           x len DECR 2
2848       FI .
2849
2850   balkenmalen           balken malen :
2851       INT VAR i;
2852       FOR i FROM 0 UPTO y len-1 REP
2853           cursor (x, y+i); out (kloetzchen)
2854           (*sh*)
2855       PER .
2856
2856   kloetzchen           kloetzchen : IF mark size > 0 THEN ""15""14"" ELSE ""15"" "14"" FI .
2857
2858   aktuellenzustandretten aktuellen zustand retten :
2859       IF  actual editor > 0
2860       THEN dateizustand retten;
2861           editstack (actual editor).feldstatus := feldstatus;
2862           editstack (actual editor).bildstatus := bildstatus;
2863           einrueckstack (actual editor) := alte einrueckposition
2864       FI .
2865
2866   neuenzustandsetzen   neuen zustand setzen :
2867       FRANGE VAR frange;
2868       feldstatus := FELDSTATUS :
2869           (1, 1, x-1, 0, 1, 0, x len-2, 0, FALSE, TRUE, access, "");
2870       bildstatus := BILDSTATUS :
2871           (x len-2, x len-2, y, y len-1, y len-1, ueberschrift, bild,
2872           0, 0, 1, 0, 0, FALSE, FALSE, "", "", "", "", "", frange, new
2873           file);
2874       alte einrueckposition := 1;
2875       dateizustand holen;
2876       ueberschrift initialisieren .

```

```

2877      neueseditorbildzeigen |neues editorbild zeigen :
2878      | ueberschrift zeigen; fenster zeigen
2879      |END PROC open editor;
2880      |

2881      openeditor ..... |PROC open editor (INT CONST i) :
2882      |   IF  i < 1 OR i > max used editor
2883      |   THEN errorstop ("Editor nicht eroeffnet")
2884      |   ELIF actual editor <> i
2885      |   THEN switch editor
2886      |   FI .
2887      |

2888      switcheditor      |switch editor :
2889      |   aktuellen zustand retten;
2890      |   actual editor := i;
2891      |   neuen zustand setzen;
2892      |   IF  kein platz mehr fuer restfenster
2893      |   THEN eingeschachtelte editoren vergessen;
2894      |         bildeinschraenkung aufheben
2895      |   ELSE neu (nix, nix)
2896      |   FI .
2897      |

2898      aktuellenzustandretten |aktuellen zustand retten :
2899      |   IF  actual editor > 0
2900      |   THEN editstack (actual editor).feldstatus := feldstatus;
2901      |         editstack (actual editor).bildstatus := bildstatus;
2902      |         einrueckstack (actual editor) := alte einrueckposition;
2903      |         dateizustand retten
2904      |   FI .
2905      |

2906      neuenzustandsetzen   |neuen zustand setzen :
2907      |   feldstatus := editstack (i).feldstatus;
2908      |   bildstatus := editstack (i).bildstatus;
2909      |   alte einrueckposition := einrueckstack (i);
2910      |   dateizustand holen .
2911      |

2912      keinplatzmehrfuerrestf |kein platz mehr fuer restfenster :
2913      |   kurze feldlaenge < 1 AND kurze bildlaenge < 1 .
2914      |

2915      eingeschachtelteeditor |eingeschachtelte editoren vergessen :
2916      |   IF  actual editor < max used editor
2917      |   THEN open editor (actual editor + 1) ;
2918      |         quit
2919      |   FI ;
2920      |   open editor (i) .
2921      |

2922      bildeinschraenkungaufh |bildeinschraenkung aufheben :
2923      |   laenge := feldlaenge;
2924      |   kurze feldlaenge := feldlaenge;
2925      |   kurze bildlaenge := bildlaenge;
2926      |   neu (ueberschrift, bild) .
2927      |END PROC open editor;
2928      |

```

```

2929 editfile .....|FILE PROC editfile :
2930                | IF actual editor = 0 OR editget modus
2931                | THEN errorstop ("Editor nicht eroeffnet")
2932                | FI ; file
2933                |END PROC editfile;
2934                |

2935 getwindow .....|PROC get window (INT VAR x, y, x size, y size) :
2936                | x := rand + 1;
2937                | y := bildrand;
2938                | x size := feldlaenge + 2;
2939                | y size := bildlaenge + 1
2940                |ENDPROC get window;
2941
2942                |{***** Zugriff auf Bildstatus
+                |*****}.
2943

2944 feldlaenge      |feldlaenge      : bildstatus.feldlaenge .
2945 kurzelfeldlaenge|kurze feldlaenge : bildstatus.kurze feldlaenge .
2946 bildrand        |bildrand        : bildstatus.bildrand .
2947 bildlaenge      |bildlaenge      : bildstatus.bildlaenge .
2948 kurzebildlaenge|kurze bildlaenge : bildstatus.kurze bildlaenge .
2949 ueberschriftbereich|ueberschriftbereich : bildstatus.ueberschriftbereich .
2950 bildbereich     |bildbereich     : bildstatus.bildbereich .
2951 ersterneusatz   |erster neusatz   : bildstatus.erster neusatz .
2952 letzterneusatz   |letzter neusatz  : bildstatus.letzter neusatz .
2953 oldzeilennr     |old zeilennr     : bildstatus.old zeilennr .
2954 oldlineno       |old lineno       : bildstatus.old lineno .
2955 oldmarklineno   |old mark lineno  : bildstatus.old mark lineno .
2956 zeileneinfuegen|zeileneinfuegen : bildstatus.zeileneinfuegen .
2957 oldlineupdate   |old line update  : bildstatus.old line update .
2958 satznrpre       |satznr pre       : bildstatus.satznr pre .
2959 ueberschriftpre|ueberschrift pre : bildstatus.ueberschrift pre .
2960 ueberschrifttext|ueberschrift text : bildstatus.ueberschrift text .
2961 ueberschriftpost|ueberschrift post : bildstatus.ueberschrift post .
2962 oldsatz         |old satz         : bildstatus.old satz .
2963 oldrange        |old range        : bildstatus.old range .
2964 file           |file            : bildstatus.file .
2965
2966                |END PACKET editor paket;

```

```

1 editorfunctions ***** PACKET editor functions  DEFINES (* FUNCTIONS
+                               052 *)
2                               (***** *) (* 17.07.85
+                               -bk- *) (* 10.09.85
3
+                               -ws- *)
4                               edit, (* 25.04.86
+                               -sh- *)
5                               show, (* 27.05.86
+                               -wk- *)
6                               U,
7                               D,
8                               T,
9                               up,
10                              down,
11                              downety,
12                              uppety,
13                              to line,
14                              PUT,
15                              GET,
16                              P,
17                              G,
18                              limit,
19                              len,
20                              eof,
21                              C,
22                              change to,
23                              CA,
24                              change all,
25                              lines,
26                              line no,
27                              col,
28                              mark,
29                              at,
30                              word,
31                              std kommando interpreter,
32                              note,
33                              note line,
34                              note edit,
35                              anything noted,
36                              note file:
37
38
39 LET marker = "^",
40 ersatzmarker = "",
41 schritt = 50,
42 file size = 4072,
43 write acc = TRUE,
44 read acc = FALSE;
45
46 LET bold = 2,
47 integer = 3,
48 string = 4,
49 end of file = 7;
50
51 LET std res = "eqvw19dpgn"9";
52
53 FILE VAR edfile;
54 BOOL VAR from scratchfile :: FALSE;
55 TEXT VAR kommandotext, tabulator, zeile;
56
57

```

```

58 stdkommandointerpreter ... | PROC std kommando interpreter (TEXT CONST taste) :
59                             | enable stop ;
60                             | edfile := editfile;
61                             | set busy indicator;
62                             | SELECT pos (std res, taste) OF
63                             | CASE 1 (*e*) : edit
64                             | CASE 2 (*q*) : quit
65                             | CASE 3 (*v*) : quit last
66                             | CASE 4 (*w*) : open editor (next editor)
67                             | CASE 5 (*1*) : toline (1); col (1)
68                             | CASE 6 (*9*) : toline (lines); col (len+1)
69                             | CASE 7 (*d*) : d case
70                             | CASE 8 (*p*) : p case
71                             | CASE 9 (*g*) : g case
72                             | CASE 10(*n*) : note edit
73                             | CASE 11(*tab*) : change tabs
74                             | OTHERWISE      : echtes kommando analysieren
75                             | END SELECT .
76
77 dcase                        | d case :
78                             | IF mark
79                             | THEN PUT ""; mark (FALSE); from scratchfile := TRUE
80                             | ELSE textzeile auf taste legen
81                             | FI .
82
83 pcase                        | p case :
84                             | IF mark
85                             |   (*sh*)
86                             | THEN IF write permission
87                             |   THEN PUT ""; push("27"12"); from scratchfile := TRUE
88                             |   ELSE out ("7")
89                             |   FI
90                             | ELSE textzeile auf taste legen
91                             | FI .
92
93 gcase                        | g case :
94                             | IF write permission
95                             |   (*sh*)
96                             | THEN IF from scratchfile
97                             |   THEN GET ""
98                             |   ELSE IF is editget
99                             |     THEN push (lernsequenz auf taste ("g")); nichts neu
100                            |   FI
101                            | ELSE out ("7")
102                            | FI .
103
104 textzelleauf tastelegen     | textzeile auf taste legen :
105                             | read record (edfile, zeile);
106                             | zeile := subtext (zeile, col);
107                             | lernsequenz auf taste legen ("g", zeile);
108                             | from scratchfile := FALSE; zeile neu .
109
110 nexteditor                  | next editor :
111                             | (aktueller editor MOD groesster editor) + 1 .

```

```

112 changetabs      |change tabs :
113                | get tabs (edfile, tabulator) ;
114                | IF pos (tabulator, marker) <> 0
115                | THEN change all (tabulator, marker, ersatzmarker)
116                | ELSE change all (tabulator, ersatzmarker, marker)
117                | FI ;
118                | put tabs (edfile, tabulator) ;
119                | ueberschrift neu .
120
121 echteskommandoanalyse |echtes kommando analysieren :
122                | kommandotext := kommando auf taste (taste);
123                | IF kommandotext = ""
124                | THEN nichts neu; LEAVE std kommando interpreter
125                | FI ;
126                | scan (kommandotext);
127                | TEXT VAR s1; INT VAR t1; next symbol (s1, t1);
128                | TEXT VAR s2; INT VAR t2; next symbol (s2, t2);
129                | IF t1 = integer AND t2 = end of file THEN toline (int (s1))
130                | ELIF t1 = string AND t2 = end of file THEN down (s1)
131                | ELIF perhaps simple up or down THEN
132                | ELIF perhaps simple changeto THEN
133                | ELSE do (kommandotext)
134                | FI .
135
136 perhapssimpleupordown |perhaps simple up or down :
137                | IF t1 = bold
138                | THEN TEXT VAR s3; INT VAR t3; next symbol (s3, t3);
139                | IF t3 <> end of file THEN FALSE
140                | ELIF s1 = "U" THEN perhaps simple up
141                | ELIF s1 = "D" THEN perhaps simple down
142                | ELSE FALSE
143                | FI
144                | ELSE FALSE
145                | FI .
146
147 perhapssimpleup      |perhaps simple up :
148                | IF t2 = string THEN up (s2); TRUE
149                | ELIF t2 = integer THEN up (int (s2)); TRUE
150                | ELSE FALSE
151                | FI .
152
153 perhapssimpledown    |perhaps simple down :
154                | IF t2 = string THEN down (s2); TRUE
155                | ELIF t2 = integer THEN down (int (s2)); TRUE
156                | ELSE FALSE
157                | FI .
158
159 perhapssimplechangeto |perhaps simple changeto :
160                | IF t1 = string AND s2 = "C" AND t3 is string AND t4 is eof
161                | THEN s1 C s3; TRUE
162                | ELSE FALSE
163                | FI .
164
165 t3isstring           |t3 is string :
166                | next symbol (s3, t3);
167                | t3 = string .

```

```

168
169 t4iseof | t4 is eof :
170 | TEXT VAR s4; INT VAR t4;
171 | next symbol (s4, t4);
172 | t4 = end of file .
173 | END PROC std kommando interpreter;
174
175
176 edit ..... | PROC edit (FILE VAR f) :
177 | enable stop;
178 | IF aktueller editor > 0
179 | (*wk*)
180 | THEN ueberschrift neu
181 | FI ;
182 | open editor (f, write acc);
183 | edit (groesster editor, std res, PROC(TEXT CONST) std kommando
184 | interpreter)
185 | END PROC edit;

186 edit ..... | PROC edit (FILE VAR f, INT CONST x, y, x size, y size) :
187 | enable stop;
188 | open editor (groesster editor + 1, f, write acc, x, y, x size, y
189 | size);
190 | edit (groesster editor, std res, PROC(TEXT CONST) std kommando
191 | interpreter)
192 | END PROC edit;

193 edit ..... | PROC edit (FILE VAR f, TEXT CONST res, PROC (TEXT CONST) kdo
194 | interpreter) :
195 | enable stop;
196 | open editor (f, write acc);
197 | edit (groesster editor, res, PROC(TEXT CONST) kdo interpreter)
198 | END PROC edit;
199

200 edit ..... | PROC edit :
201 | IF aktueller editor > 0
202 | THEN dateiname einlesen;
203 | edit (dateiname)
204 | ELSE edit (last param)
205 | FI .
206

207 dateinameeinlesen | dateiname einlesen :
208 | INT VAR x, y; get editcursor (x, y);
209 | IF x < x size - 17
210 | (*sh*)
211 | THEN cursor (x, y);
212 | out ("15"Dateiname:"14");
213 | (x size-14-x) TIMESOUT " ";
214 | (x size-14-x) TIMESOUT ""8"";

```

```

214          TEXT VAR dateiname := std;
215          editget (dateiname);
216          trailing blanks entfernen;
217          quotes entfernen
218      ELSE errorstop ("Fenster zu klein")
219      FI .
220
221  trailingblanksentferne trailing blanks entfernen:
222      INT VAR i := LENGTH dateiname;
223      WHILE (dateiname SUB i) = " " REP i DECR 1 PER;
224      dateiname := subtext (dateiname, 1, i) .
225
226  quotesentfernen  quotes entfernen :
227      IF  (dateiname SUB 1) = "" AND (dateiname SUB LENGTH dateiname)
+          = ""
228      THEN dateiname := subtext (dateiname, 2, LENGTH dateiname - 1)
229      FI .
230      END PROC edit;
231
232
233  edit ..... PROC edit (TEXT CONST filename) :
234      IF  filename <> ""
235      THEN edit named file
236      ELSE errorstop ("Name ungueltig")
237      FI .
238
239  editnamedfile  edit named file :
240      last param (filename);
241      IF  exists (filename) COR yes ("" + filename + "" neu
+          einrichten")
242      THEN IF aktueller editor > @ THEN ueberschrift neu FI;
+          (*sh*)
243      FILE VAR f := sequential file (modify, filename);
244      headline (f, filename); edit (f); last param (filename)
245      ELSE errorstop ("")
246      FI .
247      END PROC edit;
248
249
250  edit ..... PROC edit (TEXT CONST filename, INT CONST x, y, x size, y size) :
251      last param (filename);
252      IF  exists (filename) COR yes ("" + filename + "" neu
+          einrichten")
253      THEN FILE VAR f := sequential file (modify, filename);
254      headline (f, filename); edit (f, x, y, x size, y size);
255      last param (filename)
256      ELSE errorstop ("")
257      FI
258      END PROC edit;
259
260

```

```

261 edit .....|PROC edit (INT CONST i) :
262           |  edit (i, std res, PROC (TEXT CONST) std kommando interpreter)
263           |END PROC edit;
264           |
265           |

266 show .....|PROC show (FILE VAR f) :
267           |  enable stop;
268           |  open editor (f, read acc);
269           |  edit(groesster editor, std res, PROC(TEXT CONST) std kommando
+           |    interpreter);
270           |END PROC show;
271           |
272           |

273 show .....|PROC show (TEXT CONST filename) :
+           |  (*sh*)
274           |  last param (filename);
275           |  IF exists (filename)
276           |  THEN FILE VAR f := sequential file (modify, filename);
277           |  show (f); last param (filename)
278           |  ELSE errorstop (""" + filename + "" gibt es nicht")
279           |  FI
280           |END PROC show;
281           |
282           |

283 show .....|PROC show :
284           |  show (last param)
285           |END PROC show;
286           |
287           |
288           |DATASPACE VAR local space;
289           |INT VAR zeilenoffset;
290           |TEXT VAR kopierzeile;
291           |
292           |

293 PUT .....|OP PUT (TEXT CONST filename) :
294           |  nichts neu;
295           |  IF mark
296           |  THEN markierten bereich in datei schreiben
297           |  FI .
298           |

299 markiertenbereichindat markierten bereich in datei schreiben :
300           |  disable stop;
301           |  zieldatei vorbereiten;
302           |  quelledatei oeffnen;
303           |  IF noch genugend platz in der zieldatei
+           |    (*sh*)
304           |  THEN zeilenweise kopieren
305           |  ELSE errorstop ("FILE-Ueberlauf")
306           |  FI ;
307           |  quelledatei schliessen;
308           |  zieldatei schliessen;
309           |  set busy indicator .
310           |

```

```

311   zieldateivorbereiten | zieldatei vorbereiten :
312   | FRANGE VAR ganze zieldatei;
313   | IF exists (filename) THEN forget (filename); ueberschrift neu FI;
314   | FILE VAR destination;
315   | IF filename = ""
316   | THEN forget (local space); local space := nilspace;
317   | destination := sequential file (output, local space)
318   | ELSE destination := sequential file (modify, filename) ;
319   | INT CONST groesse der zieldatei := lines (destination);
320   | (*sh*)
321   | set marked range (destination, ganze zieldatei) ;
322   | output (destination)
323   | FI .

324   quelldateioeffnen | quelldatei oeffnen :
325   | zeilenoffset := mark line no (edfile) - 1;
326   | INT CONST old line := line no, old col := col;
327   | FRANGE VAR ganze datei;
328   | set range (edfile, mark lineno (edfile), mark col (edfile), ganze
329   | datei);
330   | input (edfile) .

331   nochgenuegendplatzinde | noch genuegend platz in der zieldatei :
332   | lines + groesse der zieldatei < file size .
333   |

334   zeilenweisekopieren | zeilenweise kopieren :
335   | enable stop;
336   | satznr neu;
337   | INT VAR zeile;
338   | FOR zeile FROM 1 UPTO lines (edfile) REP
339   | getline (edfile, kopierzeile);
340   | putline (destination, kopierzeile);
341   | satznr zeigen
342   | PER .
343   |

344   quelldateischliessen | quelldatei schliessen :
345   | modify (edfile);
346   | set range (edfile, ganze datei);
347   | to line (old line);
348   | col (old col) .
349   |

350   zieldateischliessen | zieldatei schliessen :
351   | IF filename <> ""
352   | THEN INT CONST last line written := line no (destination) ;
353   | modify (destination) ;
354   | to line (destination, last line written) ;
355   | col (destination, len (destination) + 1) ;
356   | bild neu (destination) ;
357   | set range (destination, ganze zieldatei)
358   | FI .
359   | END OP PUT;
360   |
361   |

```

```

362 P ..... OP P (TEXT CONST filename) :
363         PUT filename
364         END OP P ;
365
366
367 GET ..... OP GET (TEXT CONST filename) :
+         (*sh*)
368         IF NOT write permission
369         THEN errorstop ("Schreibversuch auf 'show'-Datei")
370         FI ;
371         quelldatei oeffnen;
372         IF nicht mehr genugend platz im editfile
373         THEN quelldatei schliessen; errorstop ("FILE-Ueberlauf")
374         FI ;
375         disable stop;
376         zieldatei oeffnen;
377         zeilenweise kopieren ;
378         zieldatei schliessen;
379         quelldatei schliessen;
380         set busy indicator .
381
382     quelldateioeffnen    quelldatei oeffnen :
383                         FILE VAR source;
384                         FRANGE VAR ganze quelldatei;
385                         IF filename = ""
386                         THEN source := sequential file (input, local space)
387                         ELSE IF NOT exists (filename)
388                         THEN errorstop (""" + filename + "" gibt es nicht")
389                         FI ;
390                         source := sequential file (modify, filename);
391                         INT CONST old line := line no (source),
392                                old col := col (source);
393                         set marked range (source, ganze quelldatei);
394                         input (source)
395                         FI .
396
397     nichtmehrgenuegendplat nicht mehr genugend platz im editfile :
398                         lines (source) + lines >= file size .
399
400     zeilenweisekopieren  zeilenweise kopieren :
401                         enable stop;
402                         satznr neu;
403                         INT VAR zeile;
404                         FOR zeile FROM 1 UPTO lines (source) REP
405                         getline (source, kopierzeile);
406                         putline (edfile, kopierzeile);
407                         satznr zeigen
408                         PER .
409
410     zieldateioeffnen     zieldatei oeffnen :
411                         zeilenoffset := line no - 1;
412                         leere datei in editfile einschachteln;
413                         output (edfile) .
414

```

```

415 leeredateiineditfileei |leere datei in editfile einschachteln :
416 | INT CONST range start col := col;
417 | FRANGE VAR ganze datei;
418 | set range (edfile, line no, col, ganze datei);
419 | IF lines = 1 THEN delete record (edfile) FI .
420 |
421
422 quelldateischliessen |quelldatei schliessen :
423 | IF filename <> ""
424 | THEN modify (source);
425 | set range (source, ganze quelldatei);
426 | to line (source, old line);
427 | col (source, old col)
428 | FI .
429
430 zieldateischliessen |zieldatei schliessen :
431 | modify (edfile);
432 | to line (lines);
433 | col (range start col);
434 | set range (edfile, ganze datei);
435 | abschnitt neu (zeilenoffset + 1, lines) .
436 | END OP GET;
437 |
438
439 G ..... |OP G (TEXT CONST filename) :
440 | GET filename
441 | END OP G;
442 |
443
444 len ..... |INT PROC len :
445 | len (edfile)
446 | END PROC len;
447 |
448
449 col ..... |PROC col (INT CONST stelle) :
450 | nichts neu; col (edfile, stelle)
451 | END PROC col;
452 |
453
454 col ..... |INT PROC col :
455 | col (edfile)
456 | END PROC col;
457 |
458
459 limit ..... |PROC limit (INT CONST limit) :
460 | nichts neu; max line length (edfile, limit)
461 | END PROC limit;
462 |

```

```

463 limit .....|INT PROC limit :
464                |   max line length (edfile)
465                |END PROC limit;
466                |
467                |

468 lines .....|INT PROC lines :
469                |   lines (edfile)
470                |END PROC lines;
471                |
472                |

473 lineno .....|INT PROC line no :
474                |   line no (edfile)
475                |END PROC line no;
476                |
477                |

478 toline .....|PROC to line (INT CONST satz nr) :
479                |   satznr neu;
480                |   edfile := editfile;
481                |   IF satz nr > lines
482                |   THEN toline (edfile, lines); col (len + 1)
483                |   ELSE to line (edfile, satz nr)
484                |   FI
485                |END PROC to line;
486                |
487                |

488 T .....|OP T (INT CONST satz nr) :
489                |   to line (satz nr)
490                |END OP T;
491                |
492                |

493 down .....|PROC down (INT CONST anz) :
494                |   nichts neu; down (edfile, anz)
495                |END PROC down;
496                |
497                |

498 D .....|OP D (INT CONST anz) :
499                |   down (anz)
500                |END OP D;
501                |
502                |

503 up .....|PROC up (INT CONST anz) :
504                |   nichts neu; up (edfile, anz)
505                |END PROC up;
506                |
507                |

```

```

508 U .....|OP U (INT CONST anz) :
509         | up (anz)
510         |END OP U;
511         |
512         |

513 down .....|PROC down (TEXT CONST muster) :
514         | nichts neu;
515         | REP
516         |   down (muster, schritt - line no MOD schritt);
517         |   IF pattern found
518         |   THEN LEAVE down
519         |   ELSE satznr zeigen
520         |   FI
521         | UNTIL eof PER
522         |END PROC down;
523         |
524         |

525 D .....|OP D (TEXT CONST muster) :
526         | down (muster)
527         |END OP D;
528         |
529         |

530 down .....|PROC down (TEXT CONST muster, INT CONST anz) :
531         | nichts neu; down (edfile, muster, anz)
532         |END PROC down;
533         |
534         |

535 up .....|PROC up (TEXT CONST muster) :
536         | nichts neu;
537         | REP
538         |   up (muster, (line no - 1) MOD schritt + 1);
539         |   IF pattern found
540         |   THEN LEAVE up
541         |   ELSE satznr zeigen
542         |   FI
543         | UNTIL line no = 1 PER
544         |END PROC up;
545         |
546         |

547 U .....|OP U (TEXT CONST muster) :
548         | up (muster)
549         |END OP U;
550         |
551         |

552 up .....|PROC up (TEXT CONST muster, INT CONST anz) :
553         | nichts neu; up (edfile, muster, anz)
554         |END PROC up;
555         |
556         |

```

```

557 downety .....|PROC downety (TEXT CONST muster) :
558                |  nichts neu;
559                |  IF NOT at (muster)
560                |  THEN down (muster)
561                |  FI
562                |END PROC downety;
563                |
564                |

565 downety .....|PROC downety (TEXT CONST muster, INT CONST anz) :
566                |  nichts neu; downety (edfile, muster, anz)
567                |END PROC downety;
568                |
569                |

570 uppety .....|PROC uppety (TEXT CONST muster) :
571                |  nichts neu;
572                |  IF NOT at (muster)
573                |  THEN up (muster)
574                |  FI
575                |END PROC uppety;
576                |
577                |

578 uppety .....|PROC uppety (TEXT CONST muster, INT CONST anz) :
579                |  nichts neu; uppety (edfile, muster, anz)
580                |END PROC uppety;
581                |
582                |

583 C .....|OP C (TEXT CONST old, new) :
584                |  change to (old, new)
585                |END OP C;
586                |

587 C .....|OP C (TEXT CONST replacement) :
588                |  IF NOT write permission
589                |  (*sh*)
590                |  THEN errorstop ("Schreibversuch auf 'show'-Datei")
591                |  FI ;
592                |  IF at (edfile, match(0))
593                |  THEN zeile neu; change (edfile, matchpos(0), matchend(0),
594                |  replacement)
595                |  FI
596                |END OP C;

596 changeto .....|PROC change to (TEXT CONST old, new) :
597                |  IF NOT write permission
598                |  (*sh*)
599                |  THEN errorstop ("Schreibversuch auf 'show'-Datei")
600                |  FI ;
601                |  nichts neu;
602                |  REP
603                |  downety (old, schritt - line no MOD schritt);
604                |  IF pattern found

```

```

604          |      THEN change (edfile, matchpos(0), matchend(0), new);
605          |      col (col + LENGTH new); zeile neu;
606          |      LEAVE changeto
607          |      ELSE satznr zeigen
608          |      FI
609          |      UNTIL eof PER
610          |      END PROC change to;
611          |
612          |

```

```

613 CA .....| OP CA (TEXT CONST old, new) :
614          | change all (old, new)
615          | END OP CA;
616          |
617          |

```

```

618 changeall .....| PROC change all (TEXT CONST old, new) :
619          | WHILE NOT eof REP old C new PER
620          | END PROC change all;
621          |
622          |

```

```

623 eof .....| BOOL PROC eof :
624          | eof (edfile)
625          | END PROC eof;
626          |
627          |

```

```

628 mark .....| BOOL PROC mark :
629          | mark (edfile)
630          | END PROC mark;
631          |
632          |

```

```

633 mark .....| PROC mark (BOOL CONST mark on) :
634          | nichts neu;
635          | IF mark on
636          | THEN mark (edfile, line no, col)
637          | ELSE mark (edfile, 0, 0)
638          | FI
639          | END PROC mark;
640          |
641          |

```

```

642 at .....| BOOL PROC at (TEXT CONST pattern) :
643          | at (edfile, pattern)
644          | END PROC at;
645          |

```

```

646 word .....| TEXT PROC word :
647          | word (edfile)
648          | END PROC word;
649          |
650          |

```

```

651 word .....|TEXT PROC word (TEXT CONST sep) :
652           | word (edfile, sep)
653           |END PROC word;
654           |
655           |

```

```

656 word .....|TEXT PROC word (INT CONST len) :
657           | word (edfile, len)
658           |END PROC word;
659           |
660           |
661           |LET no access = 0,
662           |     edit access = 1,
663           |     output access = 2;
664           |
665           |INT VAR last note file mode;
666           |FILE VAR notebook;
667           |INITFLAG VAR this packet := FALSE;
668           |DATASPACE VAR note ds;
669           |
670           |

```

```

671 note .....|PROC note (TEXT CONST text) :
672           | access note file (output access);
673           | write (notebook, text)
674           |END PROC note;
675           |
676           |

```

```

677 note .....|PROC note (INT CONST number) :
678           | access note file (output access);
679           | put (notebook, number)
680           |END PROC note;
681           |
682           |

```

```

683 noteline .....|PROC note line :
684           | access note file (output access);
685           | line (notebook)
686           |END PROC note line;
687           |
688           |

```

```

689 anythingnoted .....|BOOL PROC anything noted :
690           | access note file (no access);
691           | last note file mode = output access
692           |END PROC anything noted;
693           |
694           |

```

```

695 notefile .....|FILE PROC note file :
696           | access note file (output access);
697           | notebook
698           |END PROC note file;
699           |
700           |

```

```

701 noteedit .....|PROC note edit (FILE VAR context) :
+                |  (*sh*)
702                |  access note file (edit access);
703                |  make notebook erasable;
704                |  IF aktueller editor = 0
705                |  THEN open editor (1, context, write acc, 1, 1, x size - 1, y
+                |   size)
706                |  FI ;
707                |  get window size;
708                |  IF window large enough
709                |  THEN include note editor;
710                |   edit (aktueller editor-1, aktueller editor, aktueller
+                |   editor-1,
711                |   std res, PROC (TEXT CONST) std kommando interpreter)
712                |  FI .
713
714 getwindowsize  |get window size :
715                |  INT VAR x, y, windows x size, windows y size;
716                |  get window (x, y, windows x size, windows y size) .
717
718 windowlargeenough |window large enough :
719                |  windows y size > 4 .
720
721 includenoteeditor |include note editor :
722                |  open editor (aktueller editor + 1, notebook, write acc,
723                |   x, y + (windows y size + 1) DIV 2,
724                |   windows x size, windows y size DIV 2) .
725
726 makenotebookerasable |make notebook erasable :
727                |  last note file mode := edit access .
728                |  END PROC note edit;
729
730
731 noteedit .....|PROC note edit :
732                |  access note file (edit access);
733                |  make notebook erasable;
734                |  edit (notebook) .
735
736 makenotebookerasable |make notebook erasable :
737                |  last note file mode := edit access .
738                |  END PROC note edit;
739
740
741 accessnotefile .....|PROC access note file (INT CONST new mode) :
742                |  disable stop;
743                |  initialize note ds if necessary;
744                |  IF last note file mode < new mode
745                |  THEN forget (note ds);
746                |   note ds := nilspace;
747                |   notebook := sequential file (output, note ds);
748                |   headline (notebook, "notebook");
749                |   last note file mode := new mode
750                |  FI .

```

```
751 |
752 | initialize note ds if necessary :
753 |   IF NOT initialized (this packet)
754 |     THEN note ds := nilspace;
755 |       last note file mode := no access
756 |   FI .
757 | END PROC access note file;
758 |
759 | END PACKET editor functions;
```

```

1      |(* ----- VERSION 2    06.03.86 ----- *)
2  stdtransput *****|PACKET std transput DEFINES
3
4      |      sysout ,
5      |      sysin ,
6      |      put ,
7      |      putline ,
8      |      line ,
9      |      page ,
10     |      write ,
11     |      get ,
12     |      getline ,
13     |      get secret line :
14
15
16     |LET cr               = "13" ,
17     |   cr lf             = "13"10" ,
18     |   home clear       = "1"4" ,
19     |   esc               = "27" ,
20     |   rubout            = "12" ,
21     |   bell              = "7" ,
22     |   back blank back = "8" 8" ,
23     |   del line cr lf = "5"13"10" ;
24
25     |TEXT VAR number word , exit char ;
26
27     |BOOL VAR console output := TRUE, console input := TRUE ;
28
29     |FILE VAR outfile, infile ;
30     |TEXT VAR outfile name := "", infile name := "" ;
31
32
33  sysout .....|PROC sysout (TEXT CONST file name) :
34
35     |   outfile name := file name ;
36     |   IF file name = ""
37     |     THEN console output := TRUE
38     |     ELSE outfile := sequential file (output, file name) ;
39     |     console output := FALSE
40
41     |   FI
42
43     |ENDPROC sysout ;
44
45
46  sysout .....|TEXT PROC sysout :
47
48     |   outfile name
49
50     |ENDPROC sysout ;
51
52
53  sysin .....|PROC sysin (TEXT CONST file name) :
54
55     |   infile name := file name ;
56     |   IF file name = ""
57     |     THEN console input := TRUE
58     |     ELSE infile := sequential file (input, file name) ;
59     |     console input := FALSE
60
61     |   FI

```

```

57                                     |ENDPROC sysin ;
58                                     |
59 sysin .....|TEXT PROC sysin :
60             |   infile name
61             |ENDPROC sysin ;
62             |
63             |
64 put .....|PROC put (TEXT CONST word) :
65           |
66           |   IF console output
67           |     THEN out (word) ; out (" ")
68           |     ELSE put (outfile, word)
69           |   FI
70           |
71           |ENDPROC put ;
72           |
73 put .....|PROC put (INT CONST number) :
74           |
75           |   put (text (number))
76           |
77           |ENDPROC put ;
78           |
79 put .....|PROC put (REAL CONST number) :
80           |
81           |   put (text (number))
82           |
83           |ENDPROC put ;
84           |
85 putline .....|PROC putline (TEXT CONST textline) :
86             |
87             |   IF console output
88             |     THEN out (textline) ; out (cr lf)
89             |     ELSE putline (outfile, textline)
90             |   FI
91             |
92             |ENDPROC putline ;
93             |
94 line .....|PROC line :
95           |
96           |   IF console output
97           |     THEN out (cr lf)
98           |     ELSE line (outfile)
99           |   FI
100          |
101          |ENDPROC line ;
102          |

```

```

103 line .....|PROC line (INT CONST times) :
104             |
105             |   INT VAR i ;
106             |   FOR i FROM 1 UPTO times REP
107             |     line
108             |   PER
109             |
110             |ENDPROC line ;
111             |

112 page .....|PROC page :
113             |
114             |   IF console output
115             |     THEN out (home clear)
116             |   FI
117             |
118             |ENDPROC page ;
119             |

120 write .....|PROC write (TEXT CONST word) :
121             |
122             |   IF console output
123             |     THEN out (word)
124             |     ELSE write (outfile, word)
125             |   FI
126             |
127             |ENDPROC write ;
128             |
129             |

130 get .....|PROC get (TEXT VAR word) :
131             |
132             |   IF console input
133             |     THEN get from console
134             |     ELSE get (infile, word)
135             |   FI .
136             |

137 getfromconsole |get from console :
138             |   REP
139             |     word := "" ;
140             |     editget (word, " ", "", exit char) ;
141             |     echoe exit char
142             |   UNTIL word <> "" AND word <> " " PER ;
143             |   delete leading blanks .
144             |

145 deleteleadingblanks |delete leading blanks :
146             |   WHILE (word SUB 1) = " " REP
147             |     word := subtext (word,2)
148             |   PER .
149             |
150             |ENDPROC get ;
151             |

152 get .....|PROC get (TEXT VAR word, TEXT CONST separator) :
153             |
154             |   IF console input

```

```

155                                     |     THEN get from console
156                                     |     ELSE get (infile, word, separator)
157                                     |     FI .
158
159   getfromconsole                     |get from console :
160                                     |word := "" ;
161                                     |editget (word, separator, "", exit char) ;
162                                     |echoe exit char .
163                                     |
164                                     |ENDPROC get ;
165
166   echoeexitchar .....                |PROC echoe exit char :
167                                     |
168                                     |IF exit char = ""13""
169                                     |THEN out ("13"10")
170                                     |ELSE out (exit char)
171                                     |FI
172                                     |
173                                     |ENDPROC echoe exit char ;
174
175   get .....                          |PROC get (INT VAR number) :
176                                     |
177                                     |get (number word) ;
178                                     |number := int (number word)
179                                     |
180                                     |ENDPROC get ;
181
182   get .....                          |PROC get (REAL VAR number) :
183                                     |
184                                     |get (number word) ;
185                                     |number := real (number word)
186                                     |
187                                     |ENDPROC get ;
188
189   get .....                          |PROC get (TEXT VAR word, INT CONST length) :
190                                     |
191                                     |IF console input
192                                     |THEN get from console
193                                     |ELSE get (infile, word, length)
194                                     |FI .
195
196   getfromconsole                     |get from console :
197                                     |word := "" ;
198                                     |editget (word, length, exit char) ;
199                                     |echoe exit char .
200                                     |
201                                     |ENDPROC get ;
202

```

```

203 getline .....|PROC getline (TEXT VAR textline) :
204                |
205                |   IF console input
206                |       THEN get from console
207                |       ELSE getline (infile, textline)
208                |   FI .
209                |
210    getfromconsole |get from console :
211                |   textline := "" ;
212                |   editget (textline, "", "", exit char) ;
213                |   echoe exit char
214                |
215                |ENDPROC getline ;
216                |
217    getsecretline .....|PROC get secret line (TEXT VAR textline) :
218                |
219                |   TEXT VAR char ;
220                |   textline := "" ;
221                |   get start cursor position ;
222                |   get line very secret ;
223                |   IF char = esc
224                |       THEN get line little secret
225                |   FI ;
226                |   cursor to start position ;
227                |   out (del line cr lf) .
228                |
229    getlineverysecret |get line very secret :
230                |   REP
231                |       inchar (char) ;
232                |       IF char = esc OR char = cr
233                |           THEN LEAVE get line very secret
234                |       ELIF char = rubout
235                |           THEN delete last char
236                |       ELIF char >= " "
237                |           THEN textline CAT char ;
238                |               out (".")
239                |       ELSE out (bell)
240                |       FI
241                |   PER .
242                |
243    deletelastchar    |delete last char :
244                |   IF LENGTH textline = 0
245                |       THEN out (bell)
246                |       ELSE out (back blank back) ;
247                |           delete char (textline, LENGTH textline)
248                |   FI .
249                |
250    getlinelittlesecret |get line little secret :
251                |   cursor to start position ;
252                |   editget (textline, "", "", exit char) .
253                |
254    getstartcursorposition |get start cursor position :
255                |   INT VAR x, y;
256                |   get cursor (x, y) .
257                |

```

Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** std transput

```
258 cursorstartposition |cursor to start position :  
259                     | cursor (x, y) .  
260                     |  
261                     |ENDPROC get secret line ;  
262                     |  
263                     |ENDPACKET std transput ;
```

```
1
2 localmanagerpart2 *****|PACKET local manager part 2 DEFINES          (* Autor: J.Liedtke *)
3                               |                                                                    (* Stand: 25.02.85 *)
4                               | list :
5
6
7                               |TEXT VAR file name, status text;
8
9
```

```
10 list .....|PROC list :
11
12 | disable stop ;
13 | DATASPACE VAR ds := nilspace ;
14 | FILE VAR list file := sequential file (output, ds) ;
15 | headline (list file, "list") ;
16 | list (list file) ;
17 | show (list file) ;
18 | forget (ds) .
19
20 |ENDPROC list ;
21
```

```
22 list .....|PROC list (FILE VAR f) :
23
24 | enable stop ;
25 | begin list ;
26 | putline (f, "");
27 | REP
28 |   get list entry (file name, status text) ;
29 |   IF file name = ""
30 |     THEN LEAVE list
31 |   FI ;
32 |   write (f, status text + " """) ;
33 |   write (f, file name) ;
34 |   write (f, "");
35 |   line (f)
36 | PER .
37
38 |ENDPROC list ;
39
40 |ENDPACKET local manager part 2 ;
```

```

1 eumelcoderpart1 ***** PACKET eumel coder part 1 (* Autor: U.
+ Bartling *)
2 DEFINES run, run again,
3 insert,
4 prot, prot off,
5 check, check on, check off,
6 warnings, warnings on, warnings off,
7
8 help, bulletin, packets
9 :
10
11 (*****
+ *****)
12 (* *)
+ *)
13 (* EUMEL - CODER
+ *)
14 (* *)
+ *)
15 (* *)
+ *)
16 (* Zur Beschreibung des Coders siehe
+ *)
17 (* U.Bartling, J. Liedtke: EUMEL-Coder-Interface
+ *)
18 (* *)
+ *)
19 (* Stand der Dokumentation : 13.02.1986
+ *)
20 (* Stand der Implementation : 16.04.1986
+ *)
21 (* *)
+ *)
22 (* *)
+ *)
23 (*****
+ *****)
24
25
26 (***** Globale Variable *****)
27
28 TEXT VAR object name;
29
30 FILE VAR bulletin file;
31
32 INT VAR hash table pointer, nt link, permanent pointer, param link,
33 index, mode, word;
34
35 BOOL VAR found, end of params;
36

```

```

38      | (*****
39      |      | *****)
40      |      |      |
41      |      |      |      | 1. Interface zum ELAN-Compiler
42      |      |      |      | 10.04.1986 *)
43      |      |      |      |      | 1.7.5.4
44      |      |      |      |      |
45      |      |      |      |      |
46      |      |      |      |      |
47      |      |      |      |      |
48      |      |      |      |      |
49      |      |      |      |      |
50      |      |      |      |      | *****
51      |
52      | LET begin of hash table        =    0 ,
53      |     end of hash table        = 1023 ,
54      |
55      |     begin of permanent table = 22784 ,
56      |         before first pt entry = 22784 ,
57      |         first permanent entry = 22785 ,
58      |     end of permanent table   = 32767 ,
59      |
60      |     wordlength                =    1 , (* compile u n d run
61      |         time *)
62      |     two word length           =    2 ,
63      |     three word length        =    3 ,
64      |
65      |     permanent param const     = 10000 ,
66      |     permanent param var       = 20000 ,
67      |     permanent proc op         = 30000 ,
68      |     permanent type            = 30000 ,
69      |     permanent row             =    10 ,
70      |     permanent struct          =    11 ,
71      |     permanent param proc      =    12 ,
72      | (* permanent param proc end marker =    0 , *)
73      |     permanent type field      =    0 ,
74      |
75      |     ptt limit                 = 10000 ,
76      |     begin of pt minus ptt limit = 12784 ,
77      |
78      |     void                       =    0 ,
79      |     int                        =    1 ,
80      |     real                       =    2 ,
81      |     string                     =    3 ,
82      |     bool                       =    5 ,
83      |     bool result                =    6 ,
84      |     dataspace                 =    7 ,
85      |     row                        =    10 ,
86      |     struct                    =    11 ,

```

```

87          const                = 1 ,
88          var                  = 2 ,
89          (* proc              = 3 , *)
90          (* denoter          = 5 , *)
91          bold                 = 2 ,
92
93          ins                  = TRUE ,
94          no ins               = FALSE ,
95          no lst               = FALSE ,
96          sermon              = TRUE ,
97          no sermon           = FALSE ,
98
99          run again mode      = 0 ,
100         compile file mode   = 1 ,
101
102         warning message     = 2 ,
103         error message       = 4 ,
104
105         point line          = ". . . . . " ;
106
107         INT CONST permanent packet := -2 ,
108         permanent end       := -3 ;
109
110         INT VAR run again mod nr := 0 ;
111
112                                     (***** Start/Ende *****)
113
114
115

```

```

116 elan .....|PROC elan (INT CONST mode, FILE VAR source, TEXT CONST line,
117             INT VAR start module number, BOOL CONST ins, lst, rtc,
118             + ser) :
119             EXTERNAL 256
120             ENDPROC elan ;

```

(***** Hash/Namenstabelle *****)

```

123 nexthashentry |next hash entry :
124               |hash table pointer INCR wordlength .
125

```

```

126 endofhashtablereached |end of hash table reached :
127                       |hash table pointer > end of hash table .
128

```

```

129 yetanotherntentry |yet another nt entry :
130                  |nt link := cdb int (nt link) ;
131                  |nt link <> 0 . ;
132

```

```

133 declareobject .....|PROC declare object (TEXT CONST name, INT VAR nt link, pt pointer) :
134                   |EXTERNAL 10031
135                   |ENDPROC declare object ;
136

```

```

137 toobject .....|PROC to object (TEXT CONST searched object) :
138                |  hash ;
139                |  search nt entry .
140
141  hash           |hash :
142                |  hash code := 0 ;
143                |  FOR index FROM 1 UPTO LENGTH searched object REP
144                |    addmult cyclic
145                |  ENDREP .
146
147  addmultcyclic |addmult cyclic :
148                |  hash code INCR hash code ;
149                |  IF hash code > end of hash table THEN wrap around FI ;
150                |  hash code := (hash code + code (searched object SUB index)) MOD
151                |    +
152                |    1024 .
153
154  wraparound     |wrap around :
155                |  hash code DECR end of hash table .
156
157  hashcode       |hash code : nt link .
158
159  searchntentry  |search nt entry :
160                |  found := FALSE ;
161                |  WHILE yet another nt entry REP
162                |    read current entry ;
163                |    IF object name = searched object
164                |      THEN found := TRUE ;
165                |      LEAVE to object
166                |    FI
167                |  PER .
168
169  readcurrententry|read current entry :
170                |  permanent pointer := cdb int (nt link + wordlength) ;
171                |  object name := cdb text (nt link + two word length)
172                |  ENDPROC to object ;
173
174                |
175                |          (***** Permanent Tabelle *****)
176
177  nextprocedure  |next procedure :
178                |  permanent pointer := cdb int (permanent pointer) . ;
179
180  nextptparam .....|PROC next pt param :
181                |  mode := cdb int (param link) MOD ptt limit ;
182                |  param link INCR wordlength ;
183                |  IF mode = permanent row THEN skip over permanent row
184                |  ELIF mode = permanent struct THEN skip over permanent struct
185                |  FI ;
186                |  set end marker if end of list .

```

```

186 skipoverpermanentrow | skip over permanent row :
187 | param link INCR wordlength ;
188 | next pt param .
189 |
190 skipoverpermanentstruc | skip over permanent struct :
191 | REP
192 |     next pt param ;
193 |     mode := cdb int (param link)
194 |     UNTIL mode = permanent type field PER ;
195 |     param link INCR wordlength
196 | ENDPROC next pt param ;
197 |
198 setendmarkerifendoflis ... | PROC set end marker if end of list :
199 |     mode := cdb int (param link) ;
200 |     end of params := mode >= permanent proc op OR mode <= 0
201 | ENDPROC set end marker if end of list ;
202 |
203 gettypeandmode ..... | PROC get type and mode (INT VAR type) :
204 |     mode := cdb int (param link) ;
205 |     IF mode = permanent param proc THEN type of param proc
206 |                                     ELSE type of object
207 |     FI .
208 |
209 typeofparamproc | type of param proc :
210 |     param link INCR wordlength ;
211 |     get type and mode (type) ;
212 |     mode := permanent param proc .
213 |
214 typeofobject | type of object :
215 |     IF mode < 0 THEN type := 2769 + (32767 + mode) ;
216 |     mode := 0
217 |     ELSE type := mode MOD ptt limit ;
218 |     mode DECR type ;
219 |     translate type if necessary ;
220 |     translate mode if necessary
221 |     FI .
222 |
223 translatetypeifnecessa | translate type if necessary :
224 |     IF permanent row or struct THEN translate type FI .
225 |
226 translatetype | translate type :
227 |     type := param link - begin of pt minus ptt limit .
228 |
229 translatemodeifnecessa | translate mode if necessary :
230 |     IF mode = permanent param const THEN mode := const
231 |     ELIF mode = permanent param var THEN mode := var
232 |     FI .
233 |
234 permanentroworstruct | permanent row or struct :
235 |     type = permanent row OR type = permanent struct
236 | ENDPROC get type and mode ;

```

237 |
238 |
239 | (***** Allgemeine Zugriffsprozeduren *****)
240 |

241 cdbint|INT PROC cdb int (INT CONST index) :
242 | EXTERNAL 116
243 |ENDPROC cdb int ;
244 |

245 cdbtext|TEXT PROC cdb text (INT CONST index) :
246 | EXTERNAL 117
247 |ENDPROC cdb text ;
248 |

```

250 | (*****
+ |
251 | (*
+ | *)
252 | (* 10. Inspector
+ | 16.04.1986 *)
253 | (*
+ | *)
254 | (*****
+ |
255 |
256 |
257 |
258 | INT VAR line number, pattern length, packet link,
259 | begin of packet, last packet entry, indentation;
260 |
261 | TEXT VAR bulletin name, type and mode, pattern, buffer;
262 |
263 | DATASPACE VAR bulletin ds :: nilspace ;
264 |
265 | .packet name :
266 | cdb text (cdb int(packet link + wordlength) + two word length) .
267 |
268 | .within editor :
269 | aktueller editor > @ . ;
270 |

```

```

271 nameoftype ..... PROC name of type (INT CONST type) :
272 | SELECT type OF
273 | CASE void :
274 | CASE int : type and mode CAT "INT"
275 | CASE real : type and mode CAT "REAL"
276 | CASE string : type and mode CAT "TEXT"
277 | CASE bool, bool result : type and mode CAT "BOOL"
278 | CASE dataspace : type and mode CAT "DATASPACE"
279 | CASE row : type and mode CAT "ROW "
280 | CASE struct : type and mode CAT "STRUCT"
281 | OTHERWISE : complex type
282 | ENDSELECT .
283 |
284 complextype | complex type :
285 | IF type > ptt limit THEN perhaps permanent struct or row
286 | ELSE get complex type
287 | FI .
288 |
289 perhapspermanentstruct | perhaps permanent struct or row :
290 | index := type + begin of pt minus ptt limit ;
291 | mode := cdb int (index) MOD ptt limit ;
292 | IF mode = permanent row THEN get permanent row
293 | ELIF mode = permanent struct THEN get permanent struct
294 | ELSE type and mode CAT "-"
295 | FI .
296 |
297 getcomplextype | get complex type :
298 | index := type + begin of permanent table ;
299 | IF is complex type THEN get name
300 | ELSE type and mode CAT "-"
301 | FI .

```

```

302
303 iscomplextyp     is complex type :
304                  permanent type definition mode = permanent type .
305
306 getname          get name :
307                  type and mode CAT cdb text (link to type name + two word
+                  length) .
308
309 linktotypename   link to type name :
310                  cdb int (index + three word length) .
311
312 permanenttypedefinitio permanent type definition mode :
313                  cdb int (index + wordlength) .
314
315 getpermanentrow  get permanent row :
316                  INT VAR t;
317                  type and mode CAT "ROW " ;
318                  type and mode CAT text (cdb int (index + wordlength)) ;
319                  type and mode CAT " " ;
320                  param link := index + two wordlength ;
321                  get type and mode (t) ;
322                  name of type (t) .
323
324 getpermanentstruct get permanent struct :
325                  type and mode CAT "STRUCT ( ... )"
326                  ENDPROC name of type ;
327
328 help .....      PROC help (TEXT CONST proc name) :
329                  prep bulletin ;
330                  prep help ;
331                  scan (object name) ;
332                  next symbol (pattern) ;
333                  packet link := end of permanent table ;
334                  IF function = 0 THEN standard help
335                      ELSE asterisk help
336                  FI .
337
338 prephelp         prep help :
339                  object name := compress (proc name) ;
340                  INT VAR function :: 0 ;
341                  INT CONST l :: LENGTH object name ;
342                  IF l > 1 AND object name <> "***"
343                      THEN IF (object name SUB 1) = "*"
344                          THEN function INCR 2 ;
345                              delete char (object name, 1)
346                      FI ;
347                      IF (object name SUB 1) = "*"
348                          THEN function INCR 1 ;
349                              delete char (object name, 1)
350                      FI ;
351                      IF another asterisk THEN wrong function FI
352                  FI.
353

```

```

354  anotherasterisk |another asterisk :
355                  |    pos (object name, "*") <> 0 .
356
357  wrongfunction   |wrong function :
358                  |    errorstop ("unzulaessige Sternfunktion") .
359
360  standardhelp    |standard help :
361                  |    to object (pattern) ;
362                  |    IF found THEN display
363                  |        ELSE error stop ("unbekannt: " + proc name)
364                  |    FI .
365
366  display          |display :
367                  |    WHILE permanent pointer <> 0 REP
368                  |        put name of packet if necessary ;
369                  |        put specifications (pattern) ;
370                  |        next procedure
371                  |    ENDREP ;
372                  |    show bulletin file .
373
374  putnameofpacketifneces |put name of packet if necessary :
375                  |    IF new packet THEN packet link := permanent pointer ;
376                  |                    find begin of packet ;
377                  |                    writeline (2) ;
378                  |                    write packet name
379                  |    FI .
380
381  findbeginofpacket |find begin of packet :
382                  |    REP
383                  |        packet link DECR wordlength
384                  |    UNTIL begin of packet found PER .
385
386  beginofpacketfound |begin of packet found :
387                  |    cdb int (packet link) = permanent packet .
388
389  newpacket        |new packet :
390                  |    permanent pointer < packet link .
391
392  asteriskhelp     |asterisk help :
393                  |    hash table pointer := begin of hash table ;
394                  |    pattern length := LENGTH pattern - 1 ;
395                  |    REP
396                  |        list all objects in current hash table chain ;
397                  |        next hash entry
398                  |    UNTIL end of hash table reached ENDREP ;
399                  |    show bulletin file .
400
401  listallobjectsincurren |list all objects in current hash table chain :
402                  |    nt link := hash table pointer ;
403                  |    WHILE yet another nt entry REP
404                  |        permanent pointer := cdb int (nt link + wordlength) ;
405                  |        object name := cdb text (nt link + two word length) ;
406                  |        IF matching THEN into bulletin FI

```

```

407          |         PER .
408          |
409  matching | matching :
410          |         INT CONST p :: pos (object name, pattern) ;
411          |         SELECT function OF
412          |             CASE 1 : p <> @ AND p = LENGTH object name - pattern
413          |             +
414          |             length
415          |             CASE 2 : p = 1
416          |             CASE 3 : p <> @
417          |             OTHERWISE FALSE
418          |         ENDSELECT .
419
420          |
421          |
422          |
423          |
424          |
425          |
426          |
427          |
428          |
429          |
430          |
431          |
432          |
433          |
434          |
435          |
436          |
437          |
438          |
439          |
440          |
441          |
442          |
443          |
444          |
445          |
446          |
447          |
448          |
449          |
450          |
451          |
452          |
453          |
454          |
455          |
456          |

```

```

457 putparamlist ..... PROC put param list :
458                               write bulletin line ( " ( " ) ;
459                               REP
460                               INT VAR type, param mode;
461                               get type and mode ( type ) ;
462                               param mode := mode ;
463                               put type and mode ;
464                               maybe param proc ;
465                               next pt param ;
466                               IF end of params THEN write bulletin line ( " ) " ) ;
467                               LEAVE put param list
468                               FI ;
469                               write bulletin line ( " , " ) ;
470                               PER .
471
472 puttypeandmode             put type and mode :
473                               type and mode := " " ;
474                               name of type ( type ) ;
475                               type and mode CAT name of mode ;
476                               write bulletin line ( type and mode ) .
477
478 nameofmode                name of mode :
479                               IF param mode = const THEN " CONST"
480                               ELIF param mode = var THEN " VAR"
481                               ELSE " PROC"
482                               FI .
483
484 maybeparamproc           maybe param proc :
485                               IF mode = permanent param proc THEN put virtual params FI .
486
487 putvirtualparams         put virtual params :
488                               skip over result type if complex type ;
489                               IF NOT end of virtual params THEN put param list FI.
490
491 skipoverresulttypeifco  skip over result type if complex type :
492                               next pt param .
493
494 endofvirtualparams       end of virtual params :
495                               end of params
496                               ENDPROC put param list ;
497
498 nextpacket ..... PROC next packet :
499                               REP
500                               packet link INCR wordlength ;
501                               word := cdb int ( packet link ) ;
502                               IF word = permanent packet THEN true return
503                               ELIF end of permanents THEN false return
504                               FI ;
505                               ENDREP .
506
507 truereturn                true return :
508                               found := TRUE ;
509                               LEAVE next packet .

```



```

559 bulletin .....|PROC bulletin (TEXT CONST packet name) :
560                |  prep bulletin ;
561                |  scan (packet name) ;
562                |  next symbol (pattern) ;
563                |  to packet ;
564                |  IF found THEN list packet ;
565                |      show bulletin file
566                |      ELSE error stop (packet name + " ist kein Paketname")
567                |  FI .
568
569 topacket         |to packet :
570                |  last packet entry := 0 ;
571                |  get nametab link of packet name ;
572                |  packet link := before first pt entry ;
573                |  REP
574                |      packet link INCR wordlength ;
575                |      word := cdb int (packet link) ;
576                |      IF word < 0 THEN IF word = permanent packet THEN packet
577                |          found
578                |          ELIF word = permanent end THEN return
579                |          FI
580                |      FI
581                |  ENDREP .
582
583 getnametablinkofpacket |get nametab link of packet name :
584                |  to object (pattern) ;
585                |  IF NOT found THEN error stop ("unbekanntes Paket : " + packet
586                |      name) ;
587                |      LEAVE to packet
588                |  FI .
589
590 packetfound      |packet found :
591                |  IF cdb int (packet link + wordlength) = nt link
592                |      THEN last packet entry := packet link FI .
593
594 return           |return :
595                |  IF last packet entry <> 0 THEN found := TRUE ;
596                |      packet link := last packet entry
597                |      ELSE found := FALSE
598                |  FI ;
599                |  LEAVE to packet
600                |  ENDPROC bulletin ;
601
602 listpacket .....|PROC list packet :
603                |  begin of packet := packet link + word length ;
604                |  write packet name ;
605                |  find end of packet ;
606                |  run through nametab and list all packet objects .
607
608 findendofpacket  |find end of packet :
609                |  last packet entry := begin of packet ;
610                |  REP
611                |      last packet entry INCR wordlength ;
612                |      word := cdb int (last packet entry) ;
613                |  UNTIL end of packet entries PER .

```

```

612
613   endofpacketentries |end of packet entries :
614                       |word = permanent packet OR word = permanent end .
615
616   runthroughnametabandli |run through nametab and list all packet objects :
617                       |hashtable pointer := begin of hashtable ;
618                       |REP
619                       |   nt link := hashtable pointer ;
620                       |   list objects of current packet in this chain ;
621                       |   next hash entry
622                       |   UNTIL end of hashtable reached ENDREP .
623
624   listobjectsofcurrentpa |list objects of current packet in this chain :
625                       |WHILE yet another nt entry REP
626                       |   permanent pointer := cdb int (nt link + wordlength) ;
627                       |   put objects of this name
628                       |   PER .
629
630   putobjectsofthisname   |put objects of this name :
631                       |   IF there is at least one object of this name in the current
632                       |   packet
633                       |   THEN into bulletin FI .
634
635   thereisatleastoneobjec |there is at least one object of this name in the current packet :
636                       |REP
637                       |   IF permanent pointer >= begin of packet AND
638                       |   permanent pointer < last packet entry
639                       |   THEN LEAVE there is at least one object of this name
640                       |   in the current packet WITH TRUE FI ;
641                       |   next procedure
642                       |   UNTIL permanent pointer = 0 PER ;
643                       |   FALSE .
644
645   intobulletin           |into bulletin :
646                       |   object name := cdb text (nt link + two word length) ;
647                       |   object names into bulletin (BOOL PROC within packet)
648                       |ENDPROC list packet ;
649
650   withinpacket ..... |BOOL PROC within packet :
651                       |   permanent pointer >= begin of packet AND
652                       |   permanent pointer < last packet entry
653                       |ENDPROC within packet ;
654
655   objectnamesintobulleti ... |PROC object names into bulletin (BOOL PROC link ok) :
656                       |   scan (object name) ;
657                       |   next symbol (object name, mode) ;
658                       |   IF type definition THEN put type definition
659                       |   ELSE put object definitions
660                       |   FI .

```

```

661      typedefinition      |type definition :
662                          |mode = bold AND no params .
663
664      noparams             |no params :
665                          |cdb int (permanent pointer + word length) >= permanent type .
666
667      puttypedefinition    |put type definition :
668                          |put obj name (object name) ;
669                          |write bulletin line ("TYPE ") ;
670                          |writeline (1) .
671
672      putobjectdefinitions |put object definitions :
673                          |WHILE link ok REP
674                          |put specifications (object name) ;
675                          |next procedure
676                          |ENDREP
677                          |ENDPROC object names into bulletin ;
678
679      bulletin .....      |PROC bulletin :
680                          |prep bulletin ;
681                          |packet link := first permanent entry ;
682                          |REP
683                          |list packet ;
684                          |write line (4) ;
685                          |next packet
686                          |UNTIL NOT found PER ;
687                          |show bulletin file
688                          |ENDPROC bulletin ;
689
690      putobjname .....    |PROC put obj name (TEXT CONST name) :
691                          |buffer := " " ;
692                          |bulletin name := point line ;
693                          |change (bulletin name, 1, end of line or name, name) ;
694                          |buffer CAT bulletin name ;
695                          |indentation := LENGTH buffer + 1 .
696
697      endoflineorname      |end of line or name :
698                          |min (LENGTH name, LENGTH bulletin name)
699                          |ENDPROC put obj name ;
700
701      packets .....      |PROC packets :
702                          |prep bulletin ;
703                          |packet link := first permanent entry ;
704                          |REP
705                          |object name := packet name ;
706                          |put obj name (object name) ;
707                          |write line ;
708                          |next packet
709                          |UNTIL NOT found PER ;
710                          |show bulletin file
711                          |ENDPROC packets ;
712

```

```

714      | (*****
+      | *****)
715      | (*
+      | *)
716      | (*          11. ELAN Run-Interface
+      |          09.01.1986 *)
717      | (*
+      | *)
718      | (* Uebersetzen von ELAN-Programmen
+      | *)
719      | (* Bereitstellen der Ausgabeprozeduren fuer den ELAN-Compiler
+      | *)
720      | (*
+      | *)
721      | (*****
+      | *****)
722      |
723      |
724      |
725      | BOOL VAR list option      := FALSE ,
726      |         check option     := TRUE  ,
727      |         warning option  := FALSE ,
728      |         listing enabled := FALSE ;
729      |
730      | FILE VAR listing file ;
731      |
732      | TEXT VAR listing file name := "" ;
733      |
734      |

735 run .....|PROC run (TEXT CONST file name) :
736      | enable stop ;
737      | IF NOT exists (file name)
738      | THEN errorstop (""" + file name + "" gibt es nicht")
739      | FI ;
740      | last param (file name) ;
741      | run elan (file name, no ins)
742      | END PROC run;
743      |

744 run .....|PROC run :
745      | run (last param)
746      | ENDPROC run ;
747      |

748 runagain .....|PROC run again :
749      | IF run again mod nr <> 0
750      | THEN elan (run again mode, bulletin file, "", run again mod nr,
751      |          no ins, no lst, check option, no sermon)
752      | ELSE errorstop ("run again' nicht moeglich")
753      | FI
754      | ENDPROC run again ;
755      |

756 insert .....|PROC insert (TEXT CONST file name) :
757      | enable stop ;
758      | IF NOT exists (file name)
759      | THEN errorstop (""" + file name + "" gibt es nicht")

```

```

760          | FI ;
761          | last param (file name) ;
762          | run elan (file name, ins)
763          |ENDPROC insert ;
764          |

765 insert .....|PROC insert :
766          | insert (last param)
767          |ENDPROC insert ;
768          |

769 runelan .....|PROC run elan (TEXT CONST file name, BOOL CONST insert option) :
770          | FILE VAR source := sequential file (modify, file name) ;
771          | IF listing enabled
772          |   THEN open listing file
773          | FI ;
774          |
775          | disable stop ;
776          | no do again ;
777          | elan (compile file mode, source, "", run again mod nr,
778          |   insert option, list option, check option, sermon) ;
779          |
780          | IF anything noted AND command dialogue
781          |   THEN ignore halt during compiling ;
782          |   note edit (source) ;
783          |   last param (file name) ;
784          |   errorstop ("")
785          | FI .
786          |

787 ignorehaltduringcompil |ignore halt during compiling :
788          | IF is error
789          |   THEN put error ;
790          |   clear error ;
791          |   pause (5)
792          | FI .
793          |

794 openlistingfile      |open listing file :
795          | listing file := sequential file (output, listing file name) ;
796          | max line length (listing file, 130)
797          |
798          |ENDPROC run elan ;
799          |

800 outtext .....|PROC out text (TEXT CONST text, INT CONST out type) :
801          | INTERNAL 257 ;
802          | IF online
803          |   THEN out (text)
804          | FI ;
805          | IF out type = error message OR (warning option AND out type =
806          |   + warning message)
807          |   THEN note (text) ;
808          | FI ;
809          | IF listing enabled
810          |   THEN write (listing file, text)
811          | FI
812          |ENDPROC out text ;

```

```

813 outline .....|PROC out line (INT CONST out type) :
814                |INTERNAL 258 ;
815                |IF online
816                |THEN out ("13"10")
817                |FI ;
818                |IF out type = error message
819                |OR (warning option AND out type = warning message)
820                |THEN note line
821                |ELIF listing enabled
822                |THEN line (listing file)
823                |FI
824                |ENDPROC out line ;
825                |

826 prot .....|PROC prot (TEXT CONST file name) :
827                |list option := TRUE ;
828                |listing file name := file name ;
829                |listing enabled := TRUE
830                |ENDPROC prot ;
831                |

832 protoff .....|PROC prot off :
833                |list option := FALSE ;
834                |listing enabled := FALSE
835                |ENDPROC prot off ;
836                |

837 prot .....|BOOL PROC prot :
838                |list option
839                |ENDPROC prot ;
840                |

841 checkon .....|PROC check on :
842                |check option := TRUE
843                |ENDPROC check on ;
844                |

845 checkoff .....|PROC check off :
846                |check option := FALSE
847                |ENDPROC check off ;
848                |

849 check .....|BOOL PROC check :
850                |check option
851                |ENDPROC check ;
852                |

853 warnngson .....|PROC warnings on :
854                |warning option := TRUE
855                |ENDPROC warnings on ;
856                |

```

Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** eumel coder part 1

```
857 warningsoff .....|PROC warnings off :  
858                   | warning option := FALSE  
859                   |ENDPROC warnings off ;  
860                   |  
  
861 warnings .....|BOOL PROC warnings :  
862                   | warning option  
863                   |ENDPROC warnings ;  
864                   |  
865                   |ENDPACKET eumel coder part 1 ;
```

```

1          | (* ----- VERSION 2    @6.03.86 -----)
2 mathlib *****| PACKET mathlib DEFINES sqrt, **, exp, ln, log2, log10, e, pi,
3          | sin, cos, tan, sind, cosd, tand,
4          | arctan, arctand, random, initializerandom :
5
6          |
7          | LET  pii  = 3.141592653589793238462,
8          |      pi2  = 1.570796326794896619231,
9          |      pi3  = 1.047197551196597746154,
10         |      pi6  = 0.523598775598298873077,
11         |      pi4  = 1.273239544735162686151,
12         |      ln2  = 0.693147180559945309417,
13         |      lg2  = 0.301029995663981195213,
14         |      ln10 = 2.302585092994045684018,
15         |      lge  = 0.434294481903251827651,
16         |      ei   = 2.718281828459045235360,
17         |      pi180 = 57.295779513082320876798,
18         |      sqrt3 = 1.732050807568877293527,
19         |      sqr3  = 0.577350269189625764509,
20         |      sqr3p2 = 3.732050807568877293527,
21         |      sqr3m2 = 0.267949192431122706473,
22         |      sqr2  = 0.707106781186547524400;
23
24         | REAL VAR rdg::0.4711;
25
26 pi .....| REAL PROC pi:  pii    END PROC pi;
27
28 e .....| REAL PROC e :  ei    END PROC e;
29
30 ln .....| REAL PROC ln ( REAL CONST x ):
31         |     log2(x) * ln2
32         | END PROC ln;
33
34 log10 .....| REAL PROC log10( REAL CONST x ):
35         |     log2(x) * lg2
36         | END PROC log10;
37
38 log2 .....| REAL PROC log2 ( REAL CONST z ):
39         | REAL VAR t, summe::0.0, x::z;
40         | IF x=1.0 THEN 0.0
41         | ELIF x>0.0 THEN normal
42         |                    ELSE errorstop("log2: " + text(x,20)); 0.0 FI.
43
44 normal .....| normal:
45         | IF x >= 0.5 THEN normalise downwards
46         |                    ELSE normalise upwards    FI;
47         | IF x < sqr2 THEN summe := summe - 0.75; t := trans8
48         |                    ELSE summe := summe - 0.25; t := trans2 FI;
49         | summe + reihenentwicklung.

```

```

49 normalisedownwards | normalise downwards:
50 | WHILE x >= 8.0 REP x := 0.0625 * x; summe:=summe+4.0 PER;
51 | WHILE x >= 1.0 REP x := 0.5 * x; summe:=summe+1.0 PER.
52 |
53 normaliseupwards | normalise upwards:
54 | WHILE x<=0.0625 REP x := 16.0 * x; summe:=summe-4.0 PER;
55 | WHILE x<= 0.5 REP x := 2.0 * x; summe:=summe-1.0 PER.
56 |
57 trans8 | trans8: (x - 0.5946035575013605)/(x + 0.5946035575013605).
58 trans2 | trans2: (x - 0.8408964152537145)/(x + 0.8408964152537145).
59 |
60 reihenentwicklung | reihenentwicklung: x := t * t; t := 0.06405572387119384648
+ | *
61 | (((((3.465*x+4.095)*x+5.005)*x+6.435)*x+9.009)*x+15.015)*x+45.045);
62 | END PROC log2;
63 |
64 sqrt ..... | REAL PROC sqrt ( REAL CONST z ):
65 | REAL VAR y0, y1, x::z;
66 | INT VAR p :: decimal exponent(x) DIV 2;
67 | IF p <= -64 THEN 0.0
68 | ELIF x < 0.0 THEN errorstop("sqrt: " + text (x,20)); 0.0
69 | ELSE nontrivial FI.
70 |
71 nontrivial | nontrivial:
72 | set exp (decimal exponent (x) -p-p, x);
73 | IF x<10.0 THEN x := 5.3176703 - 40.760905/( 8.408065 + x )
74 | ELSE x := 16.81595 - 1288.973 /( 84.08065 + x ) FI;
75 | y0 := x;
76 | set exp (decimal exponent (x) + p, y0);
77 | y1 := 0.5 * ( y0 + z/y0 );
78 | y0 := 0.5 * ( y1 + z/y1 );
79 | y1 := 0.5 * ( y0 + z/y0 );
80 | 0.5 * ( y1 + z/y1 )
81 | END PROC sqrt;
82 |
83 exp ..... | REAL PROC exp ( REAL CONST z ):
84 | REAL VAR x::z, a::1.0; BOOL VAR negativ :: x<0.0;
85 | IF negativ THEN x := -x FI;
86 | IF x>292.42830676
87 | THEN IF NOT negativ THEN errorstop ("REAL-Ueberlauf") FI ; 0.0
88 | ELIF x<=0.0001
89 | THEN ( 0.5*z + 1.0 ) * z + 1.0
90 | ELSE approx
91 | FI.
92 |
93 approx | approx:
94 | IF x > ln10
95 | THEN x := lge*x;
96 | a := 1 √;
97 | set exp (int(x), a);
98 | x := frac(x)*ln10
99 | FI;

```

```

100 | IF x >= 2.0 THEN a := 7.389056098930650227230*a; x := x-2.0 FI;
101 | IF x >= 1.0 THEN a := 2.718281828459045235360*a; x := x-1.0 FI;
102 | IF x >= 0.5 THEN a := 1.648721270700128146848*a; x := x-0.5 FI;
103 | IF x >= 0.25 THEN a := 1.284025416687741484073*a; x := x-0.25 FI;
104 | IF x >= 0.125 THEN a := 1.133148453066826316829*a; x := x-0.125 FI;
105 | IF x >= 0.0625 THEN a := 1.064494458917859429563*a; x := x-0.0625 FI;
106 |
+ | a:=a/50.4*(((0.01*x+0.07)*x+0.42)*x+2.1)*x+8.4)*x+25.2)*x+5
+ | 0.4)*x+50.4);
107 | IF negativ THEN 1.0/a ELSE a FI .
108 |
109 | ENDPROC exp ;
110 |

```

```

111 tan ..... | REAL PROC tan (REAL CONST x):
112 | IF x < 0.0 THEN - tg( -x * pi4)
113 | ELSE tg( x * pi4) FI
114 | END PROC tan;
115 |

```

```

116 tand ..... | REAL PROC tand (REAL CONST x):
117 | IF x < 0.0 THEN - tg( -x / 45.0)
118 | ELSE tg( x / 45.0) FI
119 | END PROC tand;
120 |

```

```

121 tg ..... | REAL PROC tg (REAL CONST x ):
122 | REAL VAR q::floor(x), s::x-q; INT VAR n;
123 | q := q - floor(0.25*q) * 4.0 ;
124 | IF q < 2.0
125 | THEN IF q < 1.0
126 | THEN n:=0;
127 | ELSE n:=1; s := 1.0 - s FI
128 | ELSE IF q < 3.0
129 | THEN n:=2;
130 | ELSE n:=3; s := 1.0 - s FI
131 | FI;
132 | q := s * s;
133 | q := ((((((((-5.116186989653120e-11*q-5.608325022830701e-10)*q-
134 | 9.526170109403018e-9)*q-1.517906721393745e-7)*q-2.430939946375
+ | 515e-6)*q-
+ | 3.901461426385464e-5)*q-6.324811612385572e-4)*q-1.076606829172
+ | 646e-2)*q-
136 | 0.2617993877991508)*q+pi4);
137 |
138 | SELECT n OF
139 | CASE 0 : s/q
140 | CASE 1 : q/s
141 | CASE 2 : -q/s
142 | OTHERWISE : -s/q ENDSELECT .
143 |
144 | END PROC tg;
145 |

```

```

146 sin .....|REAL PROC sin ( REAL CONST x ):
147           | REAL VAR y, r, q;
148           | IF x < 0.0 THEN y := -x; q := 4.0 ELSE y := x; q := 0.0 FI;
149           | y := y * pi4;
150           | r := floor(y);
151           | sincos( q+r , y-r )
152           |END PROC sin;
153           |

154 sind .....|REAL PROC sind ( REAL CONST x ):
155           | REAL VAR y, r, q;
156           | IF x < 0.0 THEN y := -x; q := 4.0 ELSE y := x; q := 0.0 FI;
157           | y := y / 45.0;
158           | r := floor(y);
159           | sincos( q+r , y-r )
160           |END PROC sind;
161           |

162 cos .....|REAL PROC cos ( REAL CONST x ):
163           | REAL VAR y, q;
164           | IF x < 0.0 THEN y := -x ELSE y := x FI;
165           | y := y * pi4;
166           | q := floor(y);
167           | sincos( q+2.0, y-q )
168           |END PROC cos;
169           |

170 cosd .....|REAL PROC cosd ( REAL CONST x ):
171           | REAL VAR y, q;
172           | IF x < 0.0 THEN y := -x ELSE y := x FI;
173           | y := y / 45.0;
174           | q := floor(y);
175           | sincos( q+2.0, y-q )
176           |END PROC cosd;
177           |

178 sincos .....|REAL PROC sincos ( REAL CONST q, y ):
179           | REAL VAR r :: q - floor( 0.125*q + 0.1 ) * 8.0;
180           | IF r >= 4.0 THEN IF r >= 6.0 THEN IF r >= 7.0 THEN - sin
181           |   approx(1.0-y)
182           |   ELSE - cos approx(y)
183           |   FI
184           |   ELSE IF r >= 5.0 THEN - cos
185           |   approx(1.0-y)
186           |   ELSE - sin approx(y)
187           |   FI FI
188           |   ELSE IF r >= 2.0 THEN IF r >= 3.0 THEN sin
189           |   approx(1.0-y)
190           |   ELSE cos approx(y)
191           |   FI
192           |   ELSE IF r >= 1.0 THEN cos
193           |   approx(1.0-y)
194           |   ELSE sin approx(y)
195           |   FI FI FI
196           |END PROC sincos;

```

```

190  sinapprox .....|REAL PROC sin approx ( REAL CONST x ):
191                    |REAL VAR z::x*x;
192                    |
193                    |      x*(((0.6877101540593035e-11*z-0.1757149296873372e-8)*z+0.313
194                    |      3616216672568
195                    |
196                    |      e-6)*z-0.3657620415845891e-4)*z+0.2490394570188737e-2)*z-0.8074
197                    |      55121882e-1)*
198                    |      z+0.7853981633974483)
199                    |END PROC sin approx;

```

```

197  cosapprox .....|REAL PROC cos approx ( REAL CONST x ):
198                    |REAL VAR z::x*x;
199                    |
200                    |      (((((-0.3857761864560276e-12*z+0.115004970178141e-9)*z-0.24611
201                    |      3638267419e-7
202                    |
203                    |      )*z+0.3590860445885748e-5)*z-0.3259918869266875e-3)*z+0.1585434
204                    |      424381541e-1)
205                    |      *z-0.3084251375340425)*z+1.0
206                    |END PROC cos approx;

```

```

204  arctan .....|REAL PROC arctan ( REAL CONST y ):
205                    |REAL VAR f, z, x; BOOL VAR neg :: y < 0.0;
206                    |IF neg THEN x := -y ELSE x := y FI;
207                    |IF x>1.0 THEN f := a ELSE f := -b; neg := NOT neg FI;
208                    |z := x * x;
209                    |x := x/(((((((0.0107090276046822*z-0.01647757182108040)*z
210                    |
211                    |      +0.02177846332482151)*z-0.03019339673273880)*z+0.046560835
212                    |      61183398)*z
213                    |      -0.0880888888888888)*z+0.3333333333333333)*z+1.0);
214                    |IF neg THEN x - f ELSE f - x FI.
215
216  a .....|a:IF x>sqrt3p2 THEN x := 1.0/x; pi2 ELSE x :=
217                    |      4.0/(sqrt3+x+x*x)-sqrt3; pi3 FI.
218
219  b .....|b:IF x<sqrt3m2 THEN 0.0 ELSE x := sqrt3 - 4.0/(sqrt3+x);
220                    |      pi6 FI
221                    |END PROC arctan;

```

```

218  arctand .....|REAL PROC arctand ( REAL CONST x ):
219                    |arctan(x) * pi180
220                    |END PROC arctand;
221

```

```

222  ** .....|REAL OP ** ( REAL CONST b, e ):
223                    |IF b=0.0
224                    |      THEN IF e=0.0 THEN 1.0 ELSE 0.0 FI
225                    |      ELIF b < 0.0
226                    |      THEN errorstop("("+text(b,20)+") ** "+text(e)); (-b) ** e
227                    |      ELSE exp( e * log2( b ) * ln2 )
228                    |      FI
229                    |END OP **;

```

230

```

231 ** .....|REAL OP ** ( REAL CONST a, INT CONST b ) :
232          |
233          |   REAL VAR p := 1.0 ,
234          |           r := a ;
235          |   INT VAR  n := ABS b ,
236          |           m ;
237          |   IF ( a = 0.0 OR a = -0.0)
238          |       THEN IF b = 0
239          |           THEN 1.0
240          |           ELSE 0.0
241          |       FI
242          |   ELSE WHILE n>0 REP
243          |       m := n DIV 2 ;
244          |       IF m + m = n
245          |           THEN n := m ;
246          |           r := r*r
247          |           ELSE n DECR 1 ;
248          |           p := p*r
249          |       FI
250          |   END REP ;
251          |   IF b>0
252          |       THEN p
253          |           ELSE 1.0 / p
254          |       FI
255          |   FI .
256          |
257          |END OP ** ;
258          |

```

```

259 random .....|REAL PROC random:
260          |
261          |   rdg:=rdg+pii;rdg:=-rdg*rdg;rdg:=-rdg*rdg;rdg:=-rdg*rdg;rdg:=-frac(
262          |       +
263          |       +
264          |       rdg);rdg
265          |END PROC random;

```

```

263 initializerandom .....|PROC initializerandom ( REAL CONST z ):
264          |   rdg := frac(z)
265          |END PROC initializerandom;
266          |
267          |END PACKET mathlib;

```

```

1      | : * ----- VERSION 2      05.05.86 ----- * )
2  commandhandler ***** | PACKET command handler DEFINES      (* Autor: J.Liedtke *)
3
4      |      get command ,
5      |      analyze command ,
6      |      do command ,
7      |      command error ,
8      |      cover tracks :
9
10
11     | LET cr lf      = ""4""13""10"" ,
12     |      esc k      = ""27""k"" ,
13     |      command pre = ""4""13""      ,
14     |      command post = ""13""10""      ,
15
16     |      max command length = 2010 ,
17
18     |      tag type = 1 ,
19     |      texttype = 4 ,
20     |      eof type = 7 ;
21
22
23     | TEXT VAR command handlers own command line := "" ,
24     |      previous command line := "" ,
25     |      symbol ,
26     |      procedure ,
27     |      pattern ,
28     |      error note := "" ;
29
30     | INT VAR symbol type ;
31
32
33
34  getcommand ..... | PROC get command (TEXT CONST command text) :
35
36     |      get command (command text, command handlers own command line)
37
38     | ENDPROC get command ;
39
40
41  getcommand ..... | PROC get command (TEXT CONST command text, TEXT VAR command line) :
42
43     |      set line nr (0) ;
44     |      error protocoll ;
45     |      get command from console .
46
47
48  errorprotocoll   | error protocoll :
49
50     |      IF is error
51     |      THEN put error ;
52     |      clear error
53     |      ELSE command line := "" ;
54     |      FI .
55
56
57  getcommandfromconsole | get command from console :
58
59     |      normalize cursor ;
60     |      REP
61     |      out (command pre) ;
62     |      out (command text) ;

```

```

57         out (command post) ;
58         editget command
59         UNTIL command line <> "" PER ;
60         param position (LENGTH command line) ;
61         out (command post) .
62
63     editgetcommand      editget command :
64                         TEXT VAR exit char ;
65                         REP
66                         get cursor (x, y) ;
67                         editget (command line, max command length, x size - x,
68                             "", "k", exit char) ;
69                         ignore halt errors during editget ;
70                         break quiet if command line is too long ;
71                         IF exit char = esc k
72                             THEN cursor to begin of command input ;
73                             command line := previous command line
74                         ELIF LENGTH command line > 1
75                             THEN previous command line := command line ;
76                             LEAVE editget command
77                         ELSE LEAVE editget command
78                         FI
79                         PER .
80
81     normalizecursor    normalize cursor :
82                         INT VAR x, y;
83                         out (CrLf) ;
84                         get cursor (x, y) ;
85                         cursor (x, y) .
86
87     ignorehalterrorsduring ignore halt errors during editget :
88                         IF is error
89                             THEN clear error
90                         FI .
91
92     breakquietifcommandlin break quiet if command line is too long :
93                         IF command line is too long
94                             THEN command line := "break (quiet)"
95                         FI .
96
97     commandlineistoolong command line is too long :
98                         LENGTH command line = max command length .
99
100     cursortobeginofcommand cursor to begin of command input :
101                         out (command pre) .
102
103                         ENDPROC get command ;
104
105
106     analyzecommand ..... PROC analyze command ( TEXT CONST command list,
107                         INT CONST permitted type,
108                         INT VAR command index, number of params,
109                         TEXT VAR param 1, param 2) :
110

```

```

111 analyze command (command list, command handlers own command line,
112 permitted type, command index,
113 number of params, param 1, param 2)
114
115 ENDPROC analyze command ;
116

```

```

117 analyzecommand ..... PROC analyze command ( TEXT CONST command list, command line,
118 INT CONST permitted type,
119 INT VAR command index, number of params,
120 TEXT VAR param 1, param 2) :
121
122 error note := "" ;
123 scan (command line) ;
124 next symbol ;
125 IF symbol type <> tag type AND symbol <> "?"
126 THEN error ("Name ungueltig") ;
127 impossible command
128 ELIF pos (command list, symbol) > 0
129 THEN procedure name ;
130 parameter list pack option ;
131 nothing else in command line ;
132 decode command
133 ELSE impossible command
134 FI .
135

```

```

136 procedurename procedure name :
137 procedure := symbol ;
138 next symbol .
139

```

```

140 parameterlistpackoptio parameter list pack option :
141 number of params := 0 ;
142 param 1 := "" ;
143 param 2 := "" ;
144 IF symbol = "("
145 THEN next symbol ;
146 parameter list ;
147 IF symbol <> ")" AND error note = ""
148 THEN error (" fehlt")
149 FI
150 ELIF symbol type <> eof type
151 THEN error ("( fehlt")
152 FI .
153

```

```

154 parameterlist parameter list :
155 parameter (param 1, number of params, permitted type) ;
156 IF symbol = ","
157 THEN next symbol ;
158 parameter (param 2, number of params, permitted type) ;
159 FI .
160

```

```

161 nothingelseincommandli nothing else in command line :
162 next symbol ;
163 IF symbol <> ""
164 THEN error ("Kommando zu schwierig")
165 FI .
166

```

```

167 decodecommand |decode command :
168 |command index := index (command list, procedure, number of params)
+ |
169 |
170 impossiblecommand |impossible command :
171 |command index := 0 .
172 |
173 |ENDPROC analyze command ;
174 |

175 parameter .....|PROC parameter (TEXT VAR param, INT VAR number of params,
176 |INT CONST permitted type) :
177 |
178 |IF symbol type = text type OR symbol type = permitted type
179 |THEN param := symbol ;
180 |number of params INCR 1 ;
181 |next symbol
182 |ELSE error ("Parameter ist kein TEXT (" fehlte)");
183 |FI
184 |
185 |ENDPROC parameter ;
186 |

187 index .....|INT PROC index (TEXT CONST list, procedure, INT CONST params) :
188 |
189 |pattern := procedure ;
190 |pattern CAT ":" ;
191 |IF procedure name found
192 |THEN get colon pos ;
193 |get dot pos ;
194 |get end pos ;
195 |get command index ;
196 |get param index ;
197 |IF param index >= 0
198 |THEN command index + param index
199 |ELSE - command index
200 |FI
201 |ELSE 0
202 |FI .
203 |

204 procedurenamefound |procedure name found :
205 |INT VAR index pos := pos (list, pattern) ;
206 |WHILE index pos > 0 REP
207 |IF index pos = 1 COR (list SUB index pos - 1) <= "9"
208 |THEN LEAVE procedure name found WITH TRUE
209 |FI ;
210 |index pos := pos (list, pattern, index pos + 1)
211 |PER ;
212 |FALSE .
213 |

214 getparamindex |get param index :
215 |INT CONST param index :=
216 |pos (list, text (params), dot pos, end pos) - dot pos -
+ |
217 |1 .

```

```

218  getcommandindex |get command index :
219                  |  INT CONST command index :=
220                  |      int ( subtext (list, colon pos + 1, dot pos - 1) ) .
221                  |
222  getcolonpos      |get colon pos :
223                  |  INT CONST colon pos := pos (list, ":", index pos) .
224                  |
225  getdotpos        |get dot pos :
226                  |  INT CONST dot pos := pos (list, ".", index pos) .
227                  |
228  getendpos        |get end pos :
229                  |  INT CONST end pos := dot pos + 4 .
230                  |
231                  |ENDPROC index ;
232                  |
233  docommand ..... |PROC do command :
234                  |
235                  |  do (command handlers own command line)
236                  |
237                  |ENDPROC do command ;
238                  |
239  error .....     |PROC error (TEXT CONST message) :
240                  |
241                  |  error note := message ;
242                  |  scan ("") ;
243                  |  procedure := "-"
244                  |
245                  |ENDPROC error ;
246                  |
247  commanderror ..... |PROC command error :
248                  |
249                  |  disable stop ;
250                  |  IF error note <> ""
251                  |      THEN errorstop (error note) ;
252                  |      error note := ""
253                  |  FI ;
254                  |  enable stop
255                  |
256                  |ENDPROC command error ;
257                  |
258                  |
259  nextsymbol ..... |PROC next symbol :
260                  |
261                  |  next symbol (symbol, symbol type)
262                  |
263                  |ENDPROC next symbol ;
264                  |
265                  |

```

```
266 covertracks .....|PROC cover tracks :
267
268 | cover tracks (command handlers own command line) ;
269 | cover tracks (previous command line) ;
270 | erase buffers of compiler and do packet .
271
272 | erasebuffersofcompiler |erase buffers of compiler and do packet :
273 | do (command handlers own command line) .
274
275 |ENDPROC cover tracks ;
276
```

```
277 covertracks .....|PROC cover tracks (TEXT VAR secret) :
278
279 | INT VAR i ;
280 | FOR i FROM 1 UPTO LENGTH secret REP
281 |   replace (secret, i, " ")
282 | PER ;
283 | WHILE LENGTH secret < 13 REP
284 |   secret CAT " "
285 | PER
286
287 |ENDPROC cover tracks ;
288
289 |ENDPACKET command handler ;
```

Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** advertising

```
1 advertising *****|PACKET advertising DEFINES eumel must advertise : (* Autor:
+                       |   J.Liedtke *)
2                       |
+                       |
3                       |
4 eumelmustadvertise .....|PROC eumel must advertise :
5                       |
6                       |   IF online AND channel <= 15
7                       |   THEN out ("1"4");
8                       |       cursor (22,5);
9                       |       out ("E U M E L   Version 1.8.0/S   "13"10"10"10");
10                      |   FI .
11                      |
12                      |ENDPROC eumel must advertise ;
13                      |
14                      |ENDPACKET advertising ;
```

```

1
2 taskssingle *****|PACKET tasks single DEFINES (* Autor:
+ | J.Liedtke *) (* Stand:
3
+ | @1.06.84 *)
4 | TASK ,
5 | := ,
6 | = ,
7 | niltask ,
8 | is niltask ,
9 | myself ,
10 | archive ,
11 | father ,
12
13 | dataspaces ,
14 | pcb ,
15 | status ,
16 | channel ,
17 | clock ,
18 | storage ,
19 | continue :
20
21
22 |LET nil = 0 ,
23
24 | hex ff = 255 ,
25 | hex 7f00 = 32512 ,
26
27 | channel field = 4 ,
28 | myself no field = 9 ,
29 | myself version field = 10 ,
30
31 | lowest ds number = 4 ,
32 | highest ds number = 255 ,
33
34 | max channel = 32 ;
35
36
37
38 |TYPE TASK = STRUCT (INT no, version) ;
39
40 |TASK CONST niltask := TASK: (0,0) ,
41 | archive := TASK: (4711,4711) ,
42 | father := archive ;
43
44 myself .....|TASK PROC myself :
45 |
46 | TASK: (pcb (myself no field), pcb (myself version field))
47 |
48 |ENDPROC myself ;
49
50
51 := .....|OP := (TASK VAR dest, TASK CONST source) :
52 |
53 | CONCR (dest) := CONCR (source)
54 |
55 |ENDOP := ;
56

```

```

57 = .....|BOOL OP = (TASK CONST left, right) :
58          |
59          |   left.no = right.no  AND  left.version = right.version
60          |
61          |ENDOP = ;
62          |

63 isniltask .....|BOOL PROC isniltask (TASK CONST t) :
64          |
65          |   t.no = 0
66          |
67          |ENDPROC isniltask ;
68          |

69 pcb .....|INT PROC pcb (TASK CONST id, INT CONST field) :
70          |
71          |   EXTERNAL 104
72          |
73          |ENDPROC pcb ;
74          |

75 status .....|INT PROC status (TASK CONST id) :
76          |
77          |   EXTERNAL 107
78          |
79          |ENDPROC status ;
80          |

81 channel .....|INT PROC channel (TASK CONST id) :
82          |
83          |   pcb (id, channel field)
84          |
85          |ENDPROC channel ;
86          |

87 clock .....|REAL PROC clock (TASK CONST id) :
88          |
89          |   EXTERNAL 106
90          |
91          |ENDPROC clock ;
92          |

93 storage .....|INT PROC storage (TASK CONST id) :
94          |
95          |   INT VAR ds number, storage sum := 0, ds size;
96          |   FOR ds number FROM lowest ds number UPTO highest ds number REP
97          |     ds size := pages (ds number, id) ;
98          |     IF ds size >= 0
99          |       THEN storage sum INCR ((ds size + 1) DIV 2)
100         |     FI
101         |     PER ;
102         |     storage sum
103         |
104         |ENDPROC storage ;
105         |

```

```

106 pages .....|INT PROC pages (INT CONST ds number, TASK CONST id) :
107              |
108              |   EXTERNAL 88
109              |
110              |ENDPROC pages ;
111              |

112 continue .....|PROC continue (INT CONST channel no) :
113              |
114              |   IF channel no > 0 AND channel no <= max channel
115              |       THEN write pcb (myself, channel field, channel no)
116              |       ELSE errorstop ("ungueltige Kanalnummer")
117              |   FI
118              |
119              |ENDPROC continue ;
120              |

121 dataspaces .....|INT PROC dataspaces :
122              |
123              |   INT VAR ds number, spaces := 0 ;
124              |   FOR ds number FROM lowest ds number UPTO highest ds number REP
125              |       IF pages (ds number, pcb (myself no field)) >= 0
126              |           THEN spaces INCR 1
127              |       FI
128              |   PER ;
129              |   spaces
130              |
131              |ENDPROC dataspaces ;
132              |

133 pages .....|INT PROC pages (INT CONST ds number, INT CONST task no) :
134              |   EXTERNAL 88
135              |ENDPROC pages ;
136              |

137 writepcb .....|PROC write pcb (TASK CONST task, INT CONST field, value) :
138              |   EXTERNAL 105
139              |ENDPROC write pcb ;
140              |
141              |ENDPACKET tasks single ;

```

```

1 fontstore ***** PACKET font store (* Autor : Rudolf
+ Ruland *)
2 (* Stand :
+ 18.02.86 *)
3 DEFINES font table,
4 list font tables,
5 list fonts,
6
7 x step conversion,
8 y step conversion,
9 on string,
10 off string,
11
12 font,
13 font exists,
14 next larger font exists,
15 next smaller font exists,
16 font lead,
17 font height,
18 font depth,
19 indentation pitch,
20 char pitch,
21 extended char pitch,
22 replacement,
23 extended replacement,
24 font string,
25 y offsets,
26 bold offset,
27 get font,
28 get replacements :
29
30
31 LET underline = 1,
32 bold = 2,
33 italics = 4,
34 reverse = 8,
35
36 first font = 1,
37 max fonts = 50,
38 max extensions = 120,
39 font table type = 3009,
40
41 FONTTABLE = STRUCT (
42
43 THESAURUS font names,
44
45 TEXT replacements, font name links,
46 extension chars, extension indexes,
47
48 ROW 4 TEXT on strings, off strings,
49
50 REAL x unit, y unit,
51
52 ROW 256 INT replacements table,
53
54 INT last font, last extension
55
56 ROW max fonts STRUCT (
57 TEXT font string, font name indexes, replacements,
58 extension chars, extension indexes, y offsets,
59 ROW 256 INT pitch table, replacements table,
60 INT indentation pitch, font lead, font height, font
+ depth,

```

```

61          next larger font, next smaller font, bold offset )
62          fonts ,
63          ROW max extensions STRUCT (
64              TEXT replacements,
65              ROW 256 INT pitch table, replacements table,
66              INT std pitch
67          )
68          extensions ,
69          );
70          INT VAR font nr, list index, last font,
71              link nr, font store replacements length;
72
73          TEXT VAR fo table := "", old font table, font name links, buffer;
74
75          THESAURUS VAR font tables, font names;
76
77          INITFLAG VAR in this task := FALSE,
78              init font ds := FALSE,
79              init ds := FALSE;
80
81          BOUND FONTTABLE VAR font store;
82
83          DATASPACE VAR font ds, ds;
84
85          (*****
86
87

```

```

88 fonttable .....|PROC font table (TEXT CONST new font table) :
89
90          disable stop;
91          get font table (new font table);
92          in this task := NOT (font table = "" OR type (font ds) <> font
93              table type);
94
95          END PROC font table;
96

```

```

97 getfonttable .....|PROC get font table (TEXT CONST new font table) :
98
99          enable stop;
100         buffer := new font table;
101         change all (buffer, " ", "");
102         IF exists (buffer) CAND type (old (buffer)) = font table type
103             THEN get font table from own task
104             ELSE errorstop ("Fonttabelle "" + buffer + "" gibt es
105                 nicht");
106
107         FI;
108
109         . get font table from own task :
110         IF NOT initialized (init ds) THEN ds := nilspace FI;
111         forget (ds); ds := old (buffer);
112         new font store;
113
114         . new font store :
115         disable stop;
116         IF NOT initialized (init font ds) THEN font ds := nilspace
117             FI;
118

```

```

115         forget (font ds);
116         font ds := ds;
117         forget (ds);
118         font store := font ds;
119         fo table := buffer;
120         font names := font store. font names;
121         font name links := font store. font name links;
122         last font := font store. last font;
123         font store replacements length := LENGTH font store.
+         replacements;
124
125     END PROC get font table;
126
127

```

```

128 fonttable ..... TEXT PROC font table :
129
130     fo table
131
132     END PROC font table;
133
134

```

```

135 listfonttables ..... PROC list font tables :
136
137     enable stop;
138     font tables := empty thesaurus;
139     font tables in own task;
140     note font tables;
141     note edit;
142
143     . font tables in own task :
144     list index := 0;
145     REP get (all, buffer, list index);
146     IF buffer = "" THEN LEAVE font tables in own task FI;
147     IF type (old (buffer)) = font table type
148         AND NOT (font tables CONTAINS buffer)
149     THEN insert (font tables, buffer) FI;
150     PER;
151
152     . note font tables :
153     list index := 0;
154     REP get (font tables, buffer, list index);
155     IF buffer = ""
156         THEN LEAVE note font tables;
157         ELSE note (buffer); note line;
158     FI;
159     PER;
160
161     END PROC list font tables;
162
163

```

```

164 listfonts ..... PROC list fonts (TEXT CONST name):
165
166     initialize if necessary;
167     disable stop;
168     old font table := font table;
169     font table (name);

```

```

170 | list fonts;
171 | font table (old font table);
172 |
173 | END PROC list fonts;
174 |
175 |
176 | listfonts ..... PROC list fonts :
177 |
178 |     enable stop;
179 |     initialize if necessary;
180 |     note font table;
181 |     FOR font nr FROM first font UPTO last font REP note font PER;
182 |     note edit;
183 |
184 |     . note font table :
185 |         note ("FONTABELLE : """); note (font table);
186 |         +
187 |         note (" x einheit = "); note (text (font store. x unit));
188 |         +
189 |         note (" y einheit = "); note (text (font store. y unit));
190 |         +
191 |         note (""); noteline;
192 |
193 |     . note font :
194 |         cout (font nr);
195 |         noteline;
196 |         note (" FONT : "); note font names;
197 |         +
198 |         note (""); noteline;
199 |         note (" einrueckbreite = "); note (text(font.
200 |         indentation pitch)); note (""); noteline;
201 |         note (" durchschuss = "); note (text(font. font
202 |         lead)); note (""); noteline;
203 |         note (" fonthoehe = "); note (text(font. font
204 |         height)); note (""); noteline;
205 |         note (" fonttiefe = "); note (text(font. font
206 |         depth)); note (""); noteline;
207 |         note (" groesserer font = """); note (next larger);
208 |         +
209 |         note (""); noteline;
210 |         note (" kleinerer font = """); note (next smaller);
211 |         +
212 |         note (""); noteline;
213 |
214 |     . font : font store. fonts (font nr)
215 |     . next larger : name (font store. font names, font. next larger
216 |     font)
217 |     . next smaller : name (font store. font names, font. next
218 |     smaller font)
219 |
220 |     . note font names :
221 |         INT VAR index;
222 |         note ("");
223 |         note (name (font names, font. font name indexes ISUB 1));
224 |         note ("");
225 |         FOR index FROM 2 UPTO LENGTH font. font name indexes DIV 2
226 |         REP note ("", "");
227 |         +
228 |         note (name (font names, font. font name indexes ISUB
229 |         index));
230 |         +
231 |         note ("");
232 |
233 |     PER;
234 |
235 | END PROC list fonts;

```

217

```

218 xstepconversion ..... INT PROC x step conversion (REAL CONST cm) :
219
220     initialize if necessary;
221     IF cm >= 0.0
222         THEN int (cm * font store. x unit + 0.5 )
223         ELSE int (cm * font store. x unit - 0.5 )
224     FI
225
226 END PROC x step conversion;
227
228

```

```

229 xstepconversion ..... REAL PROC x step conversion (INT CONST steps) :
230
231     initialize if necessary;
232     real (steps) / font store. x unit
233
234 END PROC x step conversion;
235
236

```

```

237 ystepconversion ..... INT PROC y step conversion (REAL CONST cm) :
238
239     initialize if necessary;
240     IF cm >= 0.0
241         THEN int (cm * font store. y unit + 0.5 )
242         ELSE int (cm * font store. y unit - 0.5 )
243     FI
244
245 END PROC y step conversion;
246
247

```

```

248 ystepconversion ..... REAL PROC y step conversion (INT CONST steps) :
249
250     initialize if necessary;
251     real (steps) / font store. y unit
252
253 END PROC y step conversion;
254
255

```

```

256 onstring ..... TEXT PROC on string (INT CONST modification) :
257
258     initialize if necessary;
259     SELECT modification OF
260         CASE underline : font store. on strings (1)
261         CASE bold      : font store. on strings (2)
262         CASE italics   : font store. on strings (3)
263         CASE reverse   : font store. on strings (4)
264         OTHERWISE     : errorstop ("unzulaessige Modifikation"); ""
265     END SELECT
266
267 END PROC on string;
268

```

```

269
|
270 offstring .....|TEXT PROC off string (INT CONST modification) :
271
272 | initialize if necessary;
273 | SELECT modification OF
274 |     CASE underline : font store.  off strings (1)
275 |     CASE bold      : font store.  off strings (2)
276 |     CASE italics   : font store.  off strings (3)
277 |     CASE reverse   : font store.  off strings (4)
278 |     OTHERWISE     : errorstop ("unzulaessige Modifikation"); ""
279 | END SELECT
280
281 | END PROC off string;
282
283
284 font .....|INT PROC font (TEXT CONST font name) :
285
286 | initialize if necessary;
287 | buffer := font name;
288 | change all (buffer, " ", "");
289 | INT CONST link nr := link (font names, buffer)
290 | IF link nr <> 0
291 |     THEN font name links ISUB link nr
292 |     ELSE 0
293 | FI
294
295 | END PROC font;
296
297
298 font .....|TEXT PROC font (INT CONST font number) :
299
300 | initialize if necessary;
301 | IF font number >= first font AND font number <= last font
302 |     THEN name (font names, fonts. font name indexes ISUB 1)
303 |     ELSE ""
304 | FI
305
306 | . fonts : font store. fonts (font number)
307
308 | END PROC font;
309
310
311 fontexists .....|BOOL PROC font exists (TEXT CONST font name) :
312
313 | font (font name) <> 0
314
315 | END PROC font exists;
316
317
318 nextlargerfontexists .....|BOOL PROC next larger font exists(INT CONST font number,
319 |                                     INT VAR next larger font) :
320

```

```

321                     | initialize if necessary;
322                     | IF font number >= first font AND font number <= last font
323                     |     THEN next larger font := fonts. next larger font;
324                     |             IF next larger font <> @
325                     |             THEN next larger font := font name links ISUB next
326                     |             larger font;
327                     |             next larger font <> @
328                     |             ELSE FALSE
329                     |     FI
330                     |     ELSE errorstop ("Font " + text (font number) + " gibt es
331                     |             nicht");
332                     |             FALSE
333                     |     FI
334                     |     . fonts : font store. fonts (font number)
335                     | END PROC next larger font exists;
336
337

```

```

338 nextsmallerfontexists ....| BOOL PROC next smaller font exists (INT CONST font number,
339                             |                     INT VAR next smaller font) :
340
341                     | initialize if necessary;
342                     | IF font number >= first font AND font number <= last font
343                     |     THEN next smaller font := fonts. next smaller font;
344                     |             IF next smaller font <> @
345                     |             THEN next smaller font := font name links ISUB next
346                     |             smaller font;
347                     |             next smaller font <> @
348                     |             ELSE FALSE
349                     |     FI
350                     |     ELSE errorstop ("Font " + text (font number) + " gibt es
351                     |             nicht");
352                     |             FALSE
353                     |     FI
354                     |     . fonts : font store. fonts (font number)
355                     | END PROC next smaller font exists;
356
357

```

```

358 fontlead .....| INT PROC font lead (INT CONST font number) :
359
360                     | initialize if necessary;
361                     | IF font number >= first font AND font number <= last font
362                     |     THEN fonts. font lead
363                     |     ELSE errorstop ("Font " + text (font number) + " gibt es
364                     |             nicht"); @
365                     |     FI
366                     |     . fonts : font store. fonts (font number)
367                     | END PROC font lead;
368
369
370

```

```

371 fontheight .....|INT PROC font height (INT CONST font number) :
372                  |
373                  |   initialize if necessary;
374                  |   IF font number >= first font AND font number <= last font
375                  |     THEN fonts. font height
376                  |     ELSE errorstop ("Font " + text (font number) + " gibt es
+                  |       nicht"); @
377                  |   FI
378                  |
379                  |   . fonts : font store. fonts (font number)
380
381                  |END PROC font height;
382
383

```

```

384 fontdepth .....|INT PROC font depth (INT CONST font number) :
385                  |
386                  |   initialize if necessary;
387                  |   IF font number >= first font AND font number <= last font
388                  |     THEN fonts. font depth
389                  |     ELSE errorstop ("Font " + text (font number) + " gibt es
+                  |       nicht"); @
390                  |   FI
391                  |
392                  |   . fonts : font store. fonts (font number)
393
394                  |END PROC font depth;
395
396

```

```

397 indentationpitch .....|INT PROC indentation pitch (INT CONST font number) :
398                  |
399                  |   initialize if necessary;
400                  |   IF font number >= first font AND font number <= last font
401                  |     THEN fonts. indentation pitch
402                  |     ELSE errorstop ("Font " + text (font number) + " gibt es
+                  |       nicht"); @
403                  |   FI
404                  |
405                  |   . fonts : font store. fonts (font number)
406
407                  |END PROC indentation pitch;
408
409

```

```

410 charpitch .....|INT PROC char pitch (INT CONST font number,
411                  |   TEXT CONST char ) :
412                  |
413                  |   initialize if necessary;
414                  |   IF font number >= first font AND font number <= last font
415                  |     THEN INT CONST pitch := font. pitch table (code (char SUB 1) +
+                  |       1);
416                  |     IF pitch = maxint
417                  |       THEN extended char pitch (font number, char SUB 1,
+                  |         char SUB 2)
418                  |     ELIF pitch < @
419                  |       THEN pitch XOR (-maxint-1)
420                  |     ELSE pitch
421                  |   FI

```



```

474                                     ELSE replacement text (font store. replacements)
475                                     FI
476
477                                END PROC replacement;
478
479
480 extendedreplacement ..... TEXT PROC extended replacement (INT CONST font number,
481                                     TEXT CONST esc char, char ) :
482
483                                initialize if necessary;
484                                IF font number >= first font AND font number <= last font
485                                THEN process extension replacement
486                                ELSE errorstop ("Font " + text (font number) + " gibt es
+
487                                nicht"); ""
488                                FI
489
490                                . process extension replacement :
491                                determine extension link nr;
492                                IF link nr = 0
493                                THEN char
494                                ELIF link nr > font store extension replacements length
+
495                                THEN link nr DECR font store extension replacements
496                                length;
497                                replacement text (font extension. replacements)
498                                ELSE replacement text (font store extension. replacements)
499                                FI
500
501                                . determine extension link nr :
502                                INT CONST index 1 := pos (font. extension chars, esc
+
503                                char);
504                                INT CONST index 2 := pos (font store. extension chars, esc
+
505                                char);
506                                IF index 1 <> 0
507                                THEN link nr := font extension. replacements table
+
508                                (code (char) + 1);
509                                ELIF index 2 <> 0
510                                THEN link nr := font store extension. replacements
+
511                                table (code (char) + 1);
512                                ELSE errorstop (""" + esc char + char + "" hat
513                                keine Erweiterung")
514                                FI;
515
516                                . font extension                : font store. extensions (font
+
517                                extension number)
518
519                                . font extension number        : font. extension indexes ISUB
+
520                                index 1
521
522                                . font                        : font store. fonts (font number)
+
523                                . font store extension        : font store. extensions (font
+
524                                store extension number)
525
526                                . font store extension number : font store. extension indexes
+
527                                ISUB index 2
528
529                                . font store extension replacements length :
530                                IF index 2 = 0
531                                THEN 0
532                                ELSE LENGTH font store extension. replacements

```

```

523                                     |           FI
524                                     |
525                                     |END PROC extended replacement;
526                                     |
527                                     |

528 replacementtext .....|TEXT PROC replacement text (TEXT CONST replacements) :
529                                     |
530                                     |   buffer := subtext (replacements, link nr + 1,
531                                     |                   link nr + code (replacements SUB
532 +                                     |                   link nr));
533                                     |   buffer
534                                     |END PROC replacement text;
535                                     |
536                                     |

537 fontstring .....|TEXT PROC font string (INT CONST font number) :
538                                     |
539                                     |   initialize if necessary;
540                                     |   IF font number >= first font AND font number <= last font
541                                     |       THEN fonts. font string
542                                     |           ELSE errorstop ("Font " + text (font number) + " gibt es
543 +                                     |           nicht"); ""
544                                     |   FI
545                                     |   . fonts : font store. fonts (font number)
546                                     |
547                                     |END PROC font string;
548                                     |
549                                     |

550 yoffsets .....|TEXT PROC y offsets (INT CONST font number) :
551                                     |
552                                     |   initialize if necessary;
553                                     |   IF font number >= first font AND font number <= last font
554                                     |       THEN fonts. y offsets
555                                     |           ELSE errorstop ("Font " + text (font number) + " gibt es
556 +                                     |           nicht"); ""
557                                     |   FI
558                                     |   . fonts : font store. fonts (font number)
559                                     |
560                                     |END PROC y offsets;
561                                     |
562                                     |

563 boldoffset .....|INT PROC bold offset (INT CONST font number) :
564                                     |
565                                     |   initialize if necessary;
566                                     |   IF font number >= first font AND font number <= last font
567                                     |       THEN fonts. bold offset
568                                     |           ELSE errorstop ("Font " + text (font number) + " gibt es
569 +                                     |           nicht"); 0
570                                     |   FI
571                                     |   . fonts : font store. fonts (font number)
572                                     |

```

```

573                                     |END PROC bold offset;
574                                     |
575                                     |

576 getfont .....|PROC get font (INT CONST font number,
577                                     |      INT VAR indentation pitch, font lead, font height,
+                                     |      font depth,
578                                     |      ROW 256 INT VAR pitch table ) :
579                                     |
580                                     |  initialize if necessary;
581                                     |  IF font number >= first font AND font number <= last font
582                                     |     THEN indentation pitch := fonts. indentation pitch;
583                                     |           pitch table     := fonts. pitch table;
584                                     |           font lead       := fonts. font lead;
585                                     |           font height     := fonts. font height;
586                                     |           font depth      := fonts. font depth;
587                                     |     ELSE errorstop ("Font " + text (font number) + " gibt es
+                                     |     nicht");
588                                     |  FI;
589                                     |
590                                     |  . fonts : font store. fonts (font number)
591                                     |
592                                     |END PROC get font;
593                                     |
594                                     |

595 getreplacements .....|PROC get replacements (INT CONST font number,
596                                     |      TEXT VAR replacements,
597                                     |      ROW 256 INT VAR replacements table) :
598                                     |
599                                     |  initialize if necessary;
600                                     |  IF font number >= first font AND font number <= last font
601                                     |     THEN replacements      := font store. replacements;
602                                     |           replacements     CAT fonts. replacements;
603                                     |           replacements table := fonts. replacements table;
604                                     |     ELSE errorstop ("Font " + text (font number) + " gibt es
+                                     |     nicht");
605                                     |  FI;
606                                     |
607                                     |  . fonts : font store. fonts (font number)
608                                     |
609                                     |END PROC get replacements;
610                                     |
611                                     |

612 initializeifnecessary ....|PROC initialize if necessary :
613                                     |
614                                     |  IF NOT initialized (in this task)
615                                     |     THEN IF font table = ""
616                                     |           THEN in this task := FALSE;
617                                     |           errorstop ("Fonttabelle noch nicht eingestellt");
618                                     |           ELSE font table (font table);
619                                     |     FI;
620                                     |  FI;
621                                     |
622                                     |END PROC initialize if necessary;
623                                     |
624                                     |
625                                     |END PACKET font store;

```

```

1          |(* ----- VERSION 3      17.03.86 ----- *)
2  nameset *****|PACKET name set DEFINES      (* Autor: J.Liedtke *)
3
4          |    ALL ,
5          |    SOME ,
6          |    LIKE ,
7          |    + ,
8          |    - ,
9          |    / ,
10         |    do ,
11         |    FILLBY ,
12         |    remainder ,
13
14         |    fetch ,
15         |    save ,
16         |    fetch all ,
17         |    save all ,
18         |    forget ,
19         |    erase ,
20         |    insert ,
21         |    edit :
22
23
24         |LET cr lf = ""13"10" ;
25
26         |TEXT VAR name ;
27         |DATASPACE VAR edit space ;
28
29         |THESAURUS VAR remaining thesaurus := empty thesaurus ;
30
31
32 + .....|THESAURUS OP + (THESAURUS CONST left, right) :
33
34         |    THESAURUS VAR union := left ;
35         |    INT VAR index := 0 ;
36         |    get (right, name, index) ;
37         |    WHILE name <> "" REP
38         |        IF NOT (union CONTAINS name)
39         |            THEN insert (union, name)
40         |        FI ;
41         |    get (right, name, index)
42         |    PER ;
43         |    union .
44
45         |ENDOP + ;
46
47 + .....|THESAURUS OP + (THESAURUS CONST left, TEXT CONST right) :
48
49         |    THESAURUS VAR union := left ;
50         |    IF NOT (union CONTAINS right)
51         |        THEN insert (union, right)
52         |    FI ;
53         |    union .
54
55         |ENDOP + ;
56

```

```

57 - .....|THESAURUS OP - (THESAURUS CONST left, right) :
58          |
59          |   THESAURUS VAR difference := empty thesaurus ;
60          |   INT VAR index := 0 ;
61          |   get (left, name, index) ;
62          |   WHILE name <> "" REP
63          |     IF NOT (right CONTAINS name)
64          |       THEN insert (difference, name)
65          |         FI ;
66          |     get (left, name, index)
67          |   PER ;
68          |   difference .
69          |
70          |ENDOP - ;
71          |

72 - .....|THESAURUS OP - (THESAURUS CONST left, TEXT CONST right) :
73          |
74          |   THESAURUS VAR difference := left ;
75          |   INT VAR index ;
76          |   delete (difference, right, index) ;
77          |   difference .
78          |
79          |ENDOP - ;
80          |

81 / .....|THESAURUS OP / (THESAURUS CONST left, right) :
82          |
83          |   THESAURUS VAR intersection := empty thesaurus ;
84          |   INT VAR index := 0 ;
85          |   get (left, name, index) ;
86          |   WHILE name <> "" REP
87          |     IF right CONTAINS name
88          |       THEN insert (intersection, name)
89          |         FI ;
90          |     get (left, name, index)
91          |   PER ;
92          |   intersection .
93          |
94          |ENDOP / ;
95          |

96 ALL .....|THESAURUS OP ALL (TEXT CONST file name) :
97          |
98          |   FILE VAR file := sequential file (input, file name) ;
99          |   THESAURUS VAR thesaurus := empty thesaurus ;
100         |   thesaurus FILLBY file ;
101         |   thesaurus .
102         |
103         |ENDOP ALL ;
104         |

105 SOME .....|THESAURUS OP SOME (THESAURUS CONST thesaurus) :
106          |
107          |   copy thesaurus into file ;
108          |   edit file ;
109          |   copy file into thesaurus .
110         |

```

```

111  copythesaurusintofile | copy thesaurus into file :
112                        |   forget (edit space) ;
113                        |   edit space := nilspace ;
114                        |   FILE VAR file := sequential file (output, edit space) ;
115                        |   file FILLBY thesaurus .
116
117  editfile               | edit file :
118                        |   modify (file) ;
119                        |   edit (file) .
120
121  copyfileintothesarus  | copy file into thesaurus :
122                        |   THESAURUS VAR result := empty thesaurus ;
123                        |   input (file) ;
124                        |   result FILLBY file ;
125                        |   forget (edit space) ;
126                        |   result .
127
128                        | ENDOP SOME ;
129
130  SOME .....           | THESAURUS OP SOME (TASK CONST task) :
131                        |
132                        |   SOME ALL task
133
134                        | ENDOP SOME ;
135
136  SOME .....           | THESAURUS OP SOME (TEXT CONST file name) :
137                        |
138                        |   SOME ALL file name
139
140                        | ENDOP SOME ;
141
142  LIKE .....           | THESAURUS OP LIKE (THESAURUS CONST thesaurus, TEXT CONST pattern) :
143                        |
144                        |   THESAURUS VAR result:= empty thesaurus ;
145                        |   INT VAR index:= 0 ;
146                        |   REP get (thesaurus, name, index) ;
147                        |     IF name = ""
148                        |       THEN LEAVE LIKE WITH result
149                        |     ELIF name LIKE pattern
150                        |       THEN insert (result, name)
151                        |     FI
152                        |   PER ;
153                        |   result .
154
155                        | ENDOP LIKE ;
156
157  remainder .....      | THESAURUS PROC remainder :
158                        |
159                        |   remaining thesaurus
160
161                        | ENDPROC remainder ;
162

```

```

163 do .....|PROC do (PROC (TEXT CONST) operate, THESAURUS CONST thesaurus) :
164          |
165          |   INT VAR index := 0 , operation number := 0 ;
166          |   TEXT VAR name ;
167          |
168          |   remaining thesaurus := empty thesaurus ;
169          |   disable stop ;
170          |   work off thesaurus ;
171          |   fill leftover with remainder .
172          |
173          | workoffthesaurus work off thesaurus :
174          |   REP
175          |     get (thesaurus, name, index) ;
176          |     IF name = ""
177          |       THEN LEAVE work off thesaurus
178          |     FI ;
179          |     operation number INCR 1 ;
180          |     cout (operation number) ;
181          |     execute (PROC (TEXT CONST) operate, name)
182          |   UNTIL is error ENDREP .
183          |
184          | fillleftoverwithremain fill leftover with remainder :
185          |   WHILE name <> "" REP
186          |     insert (remaining thesaurus, name) ;
187          |     get (thesaurus, name, index)
188          |   PER .
189          |
190          | ENDPROC do ;
191          |
192          |
193          |
194          |
195          |
196          |
197          |
198          |
199          |
200          |
201          |
202          |
203          |
204          |
205          |
206          |
207          |
208          |
209          |
210          |
211          |
212          |
213          |
214          |
215          |
216          |

```

```

217          |      cout (operation number) ;
218          |      execute (PROC (TEXT CONST, TASK CONST) operate, name, task)
219          |      UNTIL is error ENDREP .
220
221  filleftoverwithremain |fill leftover with remainder :
222          |      WHILE name <> "" REP
223          |          insert (remaining thesaurus, name) ;
224          |          get (thesaurus, name, index)
225          |      PER .
226
227          |      ENDPROC do ;
228
229  execute .....|PROC execute (PROC (TEXT CONST, TASK CONST) operate,
230          |      TEXT CONST name, TASK CONST task) :
231
232          |      enable stop ;
233          |      operate (name, task)
234
235          |      ENDPROC execute ;
236
237  FILLBY .....|OP FILLBY (THESAURUS VAR thesaurus, FILE VAR file) :
238
239          |      WHILE NOT eof (file) REP
240          |          getline (file, name) ;
241          |          delete trailing blanks ;
242          |          IF name <> "" CAND NOT (thesaurus CONTAINS name)
243          |              THEN insert (thesaurus, name)
244          |      FI
245          |      PER .
246
247  deletetrailingblanks |delete trailing blanks :
248          |      WHILE (name SUB LENGTH name) = "" REP
249          |          name := subtext (name, 1, LENGTH name - 1)
250          |      PER .
251
252          |      ENDOP FILLBY ;
253
254  FILLBY .....|OP FILLBY (FILE VAR file, THESAURUS CONST thesaurus) :
255
256          |      INT VAR index := 0 ;
257          |      REP
258          |          get (thesaurus, name, index) ;
259          |          IF name = ""
260          |              THEN LEAVE FILLBY
261          |      FI ;
262          |          putline (file, name)
263          |      PER .
264
265          |      ENDOP FILLBY ;
266

```

```

267  FILLBY .....|OP FILLBY (TEXT CONST file name, THESAURUS CONST thesaurus) :
268                                     |
269                                     |   FILE VAR f := sequential file (output, file name) ;
270                                     |   f FILLBY thesaurus
271                                     |
272                                     |ENDOP FILLBY ;
273                                     |
274                                     |
275                                     |

```

```

276  fetch .....|PROC fetch (THESAURUS CONST nameset) :
277                                     |
278                                     |   do (PROC (TEXT CONST) fetch, nameset)
279                                     |
280                                     |ENDPROC fetch ;
281                                     |

```

```

282  fetch .....|PROC fetch (THESAURUS CONST nameset, TASK CONST task) :
283                                     |
284                                     |   do (PROC (TEXT CONST, TASK CONST) fetch, nameset, task)
285                                     |
286                                     |ENDPROC fetch ;
287                                     |

```

```

288  save .....|PROC save (THESAURUS CONST nameset) :
289                                     |
290                                     |   do (PROC (TEXT CONST) save, nameset)
291                                     |
292                                     |ENDPROC save ;
293                                     |

```

```

294  save .....|PROC save (THESAURUS CONST nameset, TASK CONST task) :
295                                     |
296                                     |   do (PROC (TEXT CONST, TASK CONST) save, nameset, task)
297                                     |
298                                     |ENDPROC save ;
299                                     |

```

```

300  fetchall .....|PROC fetch all :
301                                     |
302                                     |   fetch all (father)
303                                     |
304                                     |ENDPROC fetch all ;
305                                     |

```

```

306  fetchall .....|PROC fetch all (TASK CONST manager) :
307                                     |
308                                     |   fetch (ALL manager, manager)
309                                     |
310                                     |ENDPROC fetch all ;
311                                     |

```

```

312  saveall .....|PROC save all :
313                                     |
314                                     |   save all (father)

```

```
315 |
316 |ENDPROC save all ;
317 |

318 saveall .....|PROC save all (TASK CONST manager) :
319 |
320 |    save (ALL myself, manager)
321 |
322 |ENDPROC save all ;
323 |

324 forget .....|PROC forget (THESAURUS CONST nameset) :
325 |
326 |    do (PROC (TEXT CONST) forget, nameset)
327 |
328 |ENDPROC forget ;
329 |

330 erase .....|PROC erase (THESAURUS CONST nameset) :
331 |
332 |    do (PROC (TEXT CONST) erase, nameset)
333 |
334 |ENDPROC erase ;
335 |

336 erase .....|PROC erase (THESAURUS CONST nameset, TASK CONST task) :
337 |
338 |    do (PROC (TEXT CONST, TASK CONST) erase, nameset, task)
339 |
340 |ENDPROC erase ;
341 |

342 insert .....|PROC insert (THESAURUS CONST nameset) :
343 |
344 |    do (PROC (TEXT CONST) insert, nameset)
345 |
346 |ENDPROC insert ;
347 |

348 edit .....|PROC edit (THESAURUS CONST nameset) :
349 |
350 |    do (PROC (TEXT CONST) edit, nameset)
351 |
352 |ENDPROC edit ;
353 |
354 |ENDPACKET name set ;
```

```

1
2 systeminfo *****|PACKET system info DEFINES                (* Autor: J.Liedtke
+                    |*)
3                    |
+                    |
4                    |   task status ,
5                    |   storage info ,
6                    |   help :
7                    |
8                    |
9                    |LET channel field = 4 ,
10                   |   prio field   = 6 ,
11                   |
12                   |   cr lf       = ""13""10"" ,
13                   |   cr          = ""13"" ,
14                   |   page       = ""1""4"" ,
15                   |   begin mark= ""15"" ,
16                   |   end mark   = ""14"" ,
17                   |   bell      = ""7"" ,
18                   |   esc       = ""27"" ;
19
20
21

```

```

22 cputimeof .....|TEXT PROC cpu time of (TASK CONST actual task) :
23
24                   |   disable stop ;
25                   |   TEXT VAR result := subtext (time (clock (actual task), 12), 1, 10)
+                   |   ;
26                   |   IF is error
27                   |       THEN clear error ;
28                   |           result := 10 * ""
29                   |   FI ;
30                   |   result
31
32                   |ENDPROC cpu time of ;
33

```

```

34 taskstatus .....|PROC task status :
35
36                   |   line ;
37                   |   put (date); put (time of day) ;
38                   |   line (2) ;
39                   |   put ("Speicher:"); put (storage (myself)); putline ("K");
40                   |   put ("CPU-Zeit:"); put (cpu time of (myself)) ; line;
41                   |   line .
42
43                   |ENDPROC task status ;
44

```

```

45 storageinfo .....|PROC storage info :
46
47                   |   INT VAR size, used ;
48                   |   storage (size, used) ;
49                   |   out (""13""10""   ) ;
50                   |   put (used) ;
51                   |   put ("K von") ;
52                   |   put (size plus reserve) ;
53                   |   putline ("K sind belegt!") .

```

```

54
55     sizeplusreserve      |size plus reserve :
56                           |  int (real (size + 24) * 64.0 / 63.0 ) .
57
58                           |ENDPROC storage info ;
59
60
61 help .....             |PROC help :
62
63                           |  IF exists ("help")
64                           |    THEN FILE VAR f := sequential file (modify, "help") ;
65                           |    help (f)
66                           |    ELSE errorstop ("""help"" gibt es nicht")
67                           |  FI .
68
69                           |ENDPROC help ;
70
71 help .....             |PROC help (FILE VAR help file) :
72
73                           |  initialize help command ;
74                           |  REP
75                           |    out (page) ;
76                           |    to paragraph ;
77                           |    show paragraph ;
78                           |    get show command
79                           |  UNTIL is quit command PER .
80
81 initializehelpcommand  |initialize help command :
82                           |  TEXT VAR
83                           |    help command := getcharety ;
84                           |  IF help command = ""
85                           |    THEN help command := "0"
86                           |  FI .
87
88 toparagraph            |to paragraph :
89                           |  col (help file, 1) ;
90                           |  to line (help file, 1) ;
91                           |  downety (help file, "*" + help command + "*") ;
92                           |  IF eof (help file)
93                           |    THEN to line (help file, 1) ;
94                           |    out (bell)
95                           |  FI .
96
97 showparagraph          |show paragraph :
98                           |  show headline ;
99                           |  WHILE NOT end of help subfile REP
100                          |    show help line
101                          |  PER ;
102                          |  show bottom line .
103
104 showheadline           |show headline :
105                          |  out (begin mark) ;
106                          |  INT CONST dots := (x size - len (help file) - 5) DIV 2 ;

```

```

107 | dots TIMESOUT "." ;
108 | exec (PROC show line, help file, 4) ;
109 | dots TIMESOUT "." ;
110 | out (end mark) ;
111 | down (help file) .
112 |
113 | showhelpline | show help line :
114 | | out (cr lf) ;
115 | | exec (PROC show line, help file, 1) ;
116 | | down (help file) .
117 |
118 | showbottomline | show bottom line :
119 | | cursor (5, y size) ;
120 | | exec (PROC show line, help file, 3) ;
121 | | out (cr) .
122 |
123 | getshowcommand | get show command :
124 | | TEXT VAR char ;
125 | | get char (char) ;
126 | | IF char = esc
127 | | THEN get char (char)
128 | | FI ;
129 | | IF char >= " "
130 | | THEN help command := char
131 | | ELSE out (bell)
132 | | FI .
133 |
134 | endofhelpsubfile | end of help subfile : pos (help file,"##",1) <> @ OR eof (help file)
135 | | .
136 | isquitcommand | is quit command : help command = "q" OR help command = "Q" .
137 |
138 | | ENDPROC help ;
139 |
140 | showline ..... | PROC show line (TEXT CONST line, INT CONST from) :
141 | |
142 | | outsubtext (line, from, x size - from)
143 | |
144 | | ENDPROC show line ;
145 | |
146 | | ENDPACKET system info ;

```

```

1      | (* ----- VERSION 2      26.05.86 ----- *)
2  singleusermonitor *****| PACKET single user monitor DEFINES      (* Autor: J.Liedtke *)
3      |
4      |      monitor ,
5      |      shutup ,
6      |      save system ,
7      |      fixpoint ,
8      |      collect garbage blocks ,
9      |      set clock ,
10     |      set date :
11     |
12     |
13     | LET command list =
14     |
15     | "edit:1.0irun:3.0irunagain:5.0insert:6.0iforget:8.0irename:10.2copy:1
+     |      1.2
16     | list      list:12.0storageinfo:13.0fetch:14.1save:15.01saveall:17.0shutup:18.0
17     | help     help:19.0 " ;
18     |
19     | LET text param type = 4 ,
20     |      main channel      = 1 ,
21     |      cr = ""13"" ,
22     |
23     |      garbage collect code = 1 ,
24     |      fixpoint code      = 2 ,
25     |      shutup code        = 4 ,
26     |      shutup and save code = 12 ;
27     |
28     |
29     | INT VAR command index , number of params , previous heap size ,
30     |      old session := session ;
31     | TEXT VAR param 1, param 2 , date text;
32     |
33     |
34     | monitor .....| PROC monitor :
35     |
36     |      monitor (PROC set up)
37     |
38     | ENDPROC monitor ;
39     |
40     | monitor .....| PROC monitor (PROC init system) :
41     |
42     |      disable stop ;
43     |      previous heap size := heap size ;
44     |      REP
45     |      continue (main channel) ;
46     |      command dialogue (TRUE) ;
47     |      sysin ("") ;
48     |      sysout ("") ;
49     |      reset editor ;
50     |      init system if necessary ;
51     |      cry if not enough storage ;
52     |      get command ("gib kommando :") ;
53     |      analyze command (command list, text param type,
54     |      command index, number of params, param1,
+     |      param2) ;

```

```

55         execute command ;
56         collect heap garbage if necessary
57         PER .
58
59     collectheapgarbageifne collect heap garbage if necessary :
60         IF heap size > previous heap size + 6
61             THEN collect heap garbage ;
62                 previous heap size := heap size
63         FI .
64
65     initsystemifnecessary init system if necessary :
66         IF session <> old session
67             THEN old session := session ;
68                 continue (main channel) ;
69                 clear error ;
70                 init system ;
71                 eumel must advertise ;
72                 set date ;
73                 storage info
74         FI .
75
76     cryifnotenoughstorage cry if not enough storage :
77         INT VAR size, used ;
78         storage (size, used) ;
79         IF used > size
80             THEN out ("?7"Speicher Engpass! Dateien loeschen!"13"10"")
81         FI .
82
83     reseteditor reset editor :
84         WHILE aktueller editor > 0 REP
85             quit
86         PER .
87
88     ENDPROC monitor ;
89
90     executecommand ..... PROC execute command :
91
92         enable stop ;
93         SELECT command index OF
94             CASE 1 : edit
95             CASE 2 : edit (param1)
96             CASE 3 : run
97             CASE 4 : run (param1)
98             CASE 5 : run again
99             CASE 6 : insert
100            CASE 7 : insert (param1)
101            CASE 8 : forget
102            CASE 9 : forget (param1)
103            CASE 10: rename (param1, param2)
104            CASE 11: copy (param1, param2)
105            CASE 12: list
106            CASE 13: storage info
107            CASE 14: fetch (param1)
108            CASE 15: save
109            CASE 16: save (param1)
110            CASE 17: save all

```

```

111          | CASE 18: shutup
112          | CASE 19: help
113          | OTHERWISE do command
114          | ENDMETHOD .
115
116          |ENDPROC execute command ;
117
118          |BOOL VAR hardware clock ok ;
119          |REAL VAR now ;
120
121 setdate .....|PROC set date :
122
123          | hardware clock ok := TRUE ;
124          | try to get date and time from hardware ;
125          | IF NOT hardware clock ok
126          |     THEN get date and time from user
127          |     FI ;
128          | define date and time .
129
130 trytogetdateandtimefro |try to get date and time from hardware :
131          | disable stop ;
132          | REAL VAR previous now ;
133          | now := 0.0 ;
134          | INT VAR try ;
135          | FOR try FROM 1 UPTO 3 WHILE hardware clock ok REP
136          |     previous now := now ;
137          |     now := date (hardwares today) + time (hardwares time)
138          | UNTIL now = previous now OR is error PER ;
139          | clear error ;
140          | enable stop .
141
142 getdateandtimefromuser |get date and time from user :
143          | line (2) ;
144          | put ( "        Bitte geben Sie das heutige Datum ein :") ;
145          | date text := date ;
146          | TEXT VAR exit char ;
147          | editget (date text, cr, "", exit char) ;
148          | now := date (date text) ;
149          | line ;
150          | put ( "        und die aktuelle Uhrzeit :") ;
151          | date text := time of day ;
152          | editget (date text, cr, "", exit char) ;
153          | now INCR time (date text) ;
154          | IF NOT last conversion ok
155          |     THEN errorstop ("Falsche Zeitangabe")
156          |     FI .
157
158 hardwarestoday        |hardwares today : calendar (3) + "." + calendar (4) + "." +
+                        |calendar (5) .
159
160 hardwarestime        |hardwares time : calendar (2) + "." + calendar (1) .
161
162 definedateandtime    |define date and time :
163          | set clock (now) .
164

```

```

165                                     |ENDPROC set date ;
166                                     |
167 calendar .....|TEXT PROC calendar (INT CONST index) :
168                                     |
169                                     |   INT VAR bcd ;
170                                     |   control (10, index, 0, bcd) ;
171                                     |   IF bcd < 0
172                                     |       THEN hardware clock ok := FALSE ; ""
173                                     |       ELSE text (low digit + 10 * high digit)
174                                     |   FI .
175                                     |
176 lowdigit                             |low digit : bcd AND 15 .
177                                     |
178 highdigit                             |high digit: (bcd AND (15*256)) DIV 256 .
179                                     |
180                                     |ENDPROC calendar ;
181                                     |
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
shutup .....|PROC shutup :
                                     |
                                     |   page ;
                                     |   cursor (32, 15) ;
                                     |   put ("bitte warten") ;
                                     |   cursor (35, 12) ;
                                     |   system operation (shutup code)
                                     |
                                     |ENDPROC shutup ;

savesystem .....|PROC save system :
                                     |
                                     |   archive ("save") ;
                                     |   IF yes ("Leere Floppy eingelegt")
                                     |       THEN
                                     |           system operation (shutup and save code) ;
                                     |       FI.
                                     |
                                     |ENDPROC save system ;

collectgarbageblocks .....|PROC collect garbage blocks :
                                     |
                                     |   system operation (garbage collect code)
                                     |
                                     |ENDPROC collect garbage blocks ;

fixpoint .....|PROC fixpoint :
                                     |
                                     |   system operation (fixpoint code)
                                     |
                                     |ENDPROC fixpoint ;

```

```
214 systemoperation .....|PROC system operation (INT CONST code) :
215                       |
216                       |   INT VAR size, used ;
217                       |   storage (size, used) ;
218                       |   IF used <= size
219                       |       THEN disable stop ;
220                       |           sys op (code) ;
221                       |           ignore start message error
222                       |       ELSE errorstop ("nicht genuegend System - Speicher vorhanden")
223                       |   FI .
224                       |
225 ignorestartmessageerro|ignore start message error :
226                       |   pause (5) ;
227                       |   clear error .
228                       |
229                       |ENDPROC system operation ;
230                       |

231 sysop .....|PROC sys op (INT CONST code) :
232                       |   EXTERNAL 90
233                       |ENDPROC sys op ;
234                       |

235 setclock .....|PROC set clock (REAL CONST time) :
236                       |   EXTERNAL 103
237                       |ENDPROC set clock ;
238                       |
239                       |ENDPACKET single user monitor ;
```


Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** ur start

```
1 |
2 |check on ;
3 |command dialogue (TRUE) ;
4 |set clock (date ("19.06.86")) ;
5 |disable stop ;
6 |save system ;
7 |REP UNTIL yes ("help") PER ;
8 |archive ("help") ;
9 |fetch ("help", archive) ;
10 |REP UNTIL yes ("dev") PER ;
11 |archive ("dev") ;
12 |fetch all (archive) ;
13 |release (archive) ;
14 |save system ;
15 |configure ;
16 |set up ;
17 |monitor ;
```

```

1      |(* ----- VERSION 11      06.03.86 -----
2 basicarchive *****|PACKET basic archive DEFINES
3
4      |archive blocks ,
5      |block number ,
6      |check read ,
7      |format archive ,
8      |read block ,
9      |read ,
10     |rewind ,
11     |search dataspace ,
12     |seek ,
13     |size ,
14     |skip dataspace ,
15     |write block ,
16     |write :
17
18     |INT VAR blocknr := 0 ,
19         rerun := 0 ,
20         page := -1 ,
21         bit word := 1 ,
22         unreadable sequence length := 0 ;
23     |INT CONST all ones :=-1 ;
24
25     |DATASPACE VAR label ds ;
26
27     |LET write normal = 0 ,
28         archive version = 1 ,
29         first page stored = 2 ,
30         dr size = 3 ,
31         first bit word = 4 ,
32     |(* write deleted data mark = 64 , *)
33         inconsistent = 90 ,
34         read error = 92 ,
35         label size = 131 ;
36
37     |BOUND STRUCT (ALIGN dummy for page1,
38         (* Page 2 begins: *)
39         ROW label size INT lab) VAR label;
40
41
42
43 blocknumber .....|INT PROC block number :
44     |block nr
45     |ENDPROC block number ;
46
47
48 seek .....|PROC seek (INT CONST block) :
49     |block nr := block
50     |ENDPROC seek ;
51
52
53 rewind .....|PROC rewind :
54     |forget (label ds);
55     |label ds := nilspace;
56     |label := label ds;
57     |block nr := 0;
58     |rerun := session

```

```

57          |END PROC rewind;
58          |
59 skipdataspace .....|PROC skip dataspace:
60          |  check rerun;
61          |  get label;
62          |  IF is error
63          |    THEN
64          |    ELIF olivetti
65          |      THEN block nr INCR label.lab (dr size+1)
66          |      ELSE block nr INCR label.lab (dr size)
67          |    FI
68          |END PROC skip dataspace;
69          |
70 read .....|PROC read (DATASPACE VAR ds):
71          |  read (ds, 30000, FALSE)
72          |ENDPROC read ;
73          |
74 read .....|PROC read (DATASPACE VAR ds, INT CONST max pages, BOOL CONST error
+          |  accept) :
75          |  enable stop ;
76          |  check rerun;
77          |  get label;
78          |  init next page;
79          |  INT VAR i ;
80          |  FOR i FROM 1 UPTO max pages REP
81          |    next page;
82          |    IF no further page THEN LEAVE read FI;
83          |    check storage ;
84          |    check rerun ;
85          |    read block ;
86          |    block nr INCR 1;
87          |  PER .
88          |
89 readblock      |read block :
90          |  disable stop ;
91          |  get external block (ds, page, block nr) ;
92          |  ignore read error if no errors accepted ;
93          |  enable stop .
94          |
95 ignorereaderrorifnoerr |ignore read error if no errors accepted :
96          |  IF is error CAND error code = read error CAND NOT error accept
97          |    THEN clear error
98          |  FI .
99          |
100 checkstorage   |check storage :
101          |  INT VAR size, used ;
102          |  storage (size, used) ;
103          |  IF used > size
104          |    THEN forget (ds) ;
105          |      ds := nilspace ;
106          |      errorstop ("Speicherengpass") ;
107          |    LEAVE read
108          |  FI .

```

```

109
110 checkrerun      |check rerun :
111                 |    IF rerun <> session
112                 |    THEN errorstop ("RERUN beim Archiv-Zugriff") ;
113                 |    LEAVE read
114                 |    FI .
115
116                 |END PROC read;
117

118 checkread .....|PROC check read :
119
120                 |enable stop ;
121                 |get label ;
122                 |INT VAR pages, i;
123                 |IF olivetti
124                 |    THEN pages := label.lab (dr size+1)
125                 |    ELSE pages := label.lab (dr size)
126                 |    FI ;
127                 |FOR i FROM 1 UPTO pages REP
128                 |    get external block (label ds, 2, block nr) ;
129                 |    block nr INCR 1
130                 |    PER .
131
132                 |ENDPROC check read ;
133

134 write .....    |PROC write (DATASPACE CONST ds):
135                 |enable stop ;
136                 |check rerun;
137                 |INT VAR label block nr := block nr;
138                 |block nr INCR 1;init label;
139                 |INT VAR page := -1,i;
140                 |FOR i FROM 1 UPTO ds pages (ds) REP
141                 |    check rerun ;
142                 |    page := next ds page(ds,page);
143                 |    put external block (ds, page, block nr) ;
144                 |    reset archive bit;
145                 |    label.lab(dr size) INCR 1;
146                 |    block nr INCR 1
147                 |    PER;
148                 |    put label.
149
150

151 initlabel      |init label:
152                 |label.lab(archive version) := 0 ;
153                 |label.lab(first page stored) := 0 ;
154                 |label.lab(dr size) := 0;
155                 |INT VAR j;
156                 |FOR j FROM first bit word UPTO label size REP
157                 |    label.lab (j) := all ones
158                 |    PER.
159

160 putlabel       |put label:
161                 |put external block (label ds, 2, label block nr).
162

```



```

215      .
216      olivetti      olivetti : label.lab (archive version) = -1.
217
218      z80archive    z80 archive : label.lab (archive version) = 0.
219
220      initnextpage  init next page:
221                      BOOL VAR no further page := false;
222                      bitword := first bit word.
223
224      checkrerun    check rerun :
225                      IF rerun <> session
226                      THEN errorstop ("RERUN beim Archiv-Zugriff")
227                      FI .
228
229      getexternalblock ..... PROC get external block (DATASPACE VAR ds, INT CONST page,
230                      INT CONST block nr):
231
232                      INT VAR error ;
233                      read block (ds, page, block nr, error) ;
234                      SELECT error OF
235                      CASE 0: read succeeded
236                      CASE 1: error stop ("Lesen unmoeglich (Archiv)")
237                      CASE 2: read failed
238                      CASE 3: error stop ("Archiv-Ueberlauf")
239                      OTHERWISE error stop ("??? (Archiv)")
240                      END SELECT .
241
242      readsucceeded  read succeeded :
243                      unreadable sequence length := 0 .
244
245      readfailed     read failed :
246                      unreadable sequence length INCR 1 ;
247                      IF unreadable sequence length >= 30
248                      THEN errorstop ("30 unlesbare Bloecke hintereinander")
249                      ELSE error stop (read error, "Lesefehler (Archiv)")
250                      FI .
251
252                      END PROC get external block;
253
254      putexternalblock ..... PROC put external block (DATASPACE CONST ds, INT CONST page,
255                      INT CONST block nr):
256                      INT VAR error;
257                      write block (ds, page, write normal, block nr, error) ;
258                      SELECT error OF
259                      CASE 0:
260                      CASE 1: error stop ("Schreiben unmoeglich (Archiv)")
261                      CASE 2: error stop ("Schreibfehler (Archiv)")
262                      CASE 3: error stop ("Archiv-Ueberlauf")
263                      OTHERWISE error stop ("??? (Archiv)")
264                      END SELECT .
265
266                      END PROC put external block;

```

```

267
268 readblock .....|PROC read block (DATASPACE VAR ds,
269                  |          INT CONST ds page no,
270                  |          INT CONST block no,
271                  |          INT VAR return code) :
272                  |  read block;
273                  |  retry if read error.
274
275 readblock        |read block:
276                  |  block in (ds, ds page no, @, block no, return code).
277
278 retryifreaderror |retry if read error:
279                  |  INT VAR retry;
280                  |  FOR retry FROM 1 UPTO 10 WHILE return code = 2 REP
281                  |    reset to block @ if fifth try;
282                  |    read block
283                  |  PER.
284
285 resettoblock@iffifthtr |reset to block @ if fifth try:
286                  |  IF retry = 5
287                  |    THEN block in (ds, ds page no, @, @, return code)
288                  |  FI.
289
290                  |END PROC read block;
291
292 writeblock .....|PROC write block (DATASPACE CONST ds,
293                  |          INT CONST ds page no,
294                  |          INT CONST mode,
295                  |          INT CONST block no,
296                  |          INT VAR return code):
297                  |  write block;
298                  |  retry if write error.
299
300 writeblock        |write block:
301                  |  block out (ds, ds page no, mode * 256, block no, return code) .
302
303 retryifwriteerror |retry if write error:
304                  |  INT VAR retry;
305                  |  FOR retry FROM 1 UPTO 10 WHILE return code = 2 REP
306                  |    reset to block @ if fifth try;
307                  |    write block
308                  |  PER.
309
310 resettoblock@iffifthtr |reset to block @ if fifth try:
311                  |  IF retry = 5
312                  |    THEN disable stop;
313                  |    DATASPACE VAR dummy ds := nilspace;
314                  |    block in (dummy ds, 2, @, @, return code);
315                  |    forget (dummy ds);
316                  |    enable stop
317                  |  FI.
318

```

```

319                                     |END PROC write block;
320                                     |

321 size .....|INT PROC size (INT CONST key) :
322                                     |
323                                     |   INT VAR return code ;
324                                     |   control (5, key, 0, return code) ;
325                                     |   return code .
326                                     |
327                                     |ENDPROC size ;
328                                     |

329 archiveblocks .....|INT PROC archive blocks :
330                                     |   size (0)
331                                     |ENDPROC archive blocks ;
332                                     |

333 searchdataspace .....|PROC search dataspace (INT VAR ds pages) :
334                                     |
335                                     |   disable stop ;
336                                     |   ds pages := -1 ;
337                                     |   INT CONST last block := archive blocks ;
338                                     |
339                                     |   WHILE block nr < last block REP
340                                     |       IF block is dataspace label
341                                     |           THEN ds pages := pages counted ;
342                                     |               LEAVE search dataspace
343                                     |       FI ;
344                                     |       block nr INCR 1
345                                     |   UNTIL is error PER .
346                                     |

347 blockisdataspacelabel |block is dataspace label :
348                                     |   look at label block ;
349                                     |   IF is error
350                                     |       THEN IF error code = read error OR error code = inconsistent
351                                     |           THEN clear error
352                                     |           FI ;
353                                     |       FALSE
354                                     |   ELSE count pages ;
355                                     |       pages counted = number of pages as label says
356                                     |   FI .
357                                     |

358 lookatlabelblock      |look at label block :
359                                     |   INT CONST
360                                     |   old block nr := block nr ;
361                                     |   get label ;
362                                     |   block nr := old block nr.
363                                     |

364 countpages            |count pages :
365                                     |   INT VAR
366                                     |   pages counted := 0 ;
367                                     |   init next page ;
368                                     |   next page ;
369                                     |   WHILE NOT no further page REP
370                                     |       pages counted INCR 1 ;
371                                     |   next page

```

```

372          | PER .
373          |
374  numberofpagesaslabels | number of pages as label says : label.lab (dr size) .
375          |
376          | ENDPROC search dataspace ;
377          |
378  formatarchive ..... | PROC format archive (INT CONST format code) :
379          |
380          | IF format is possible
381          |     THEN format
382          |     ELSE errorstop ("format' ist hier nicht implementiert")
383          |     FI .
384          |
385  formatispossible     | format is possible :
386          |     INT VAR return code ;
387          |     control (1,0,0, return code) ;
388          |     bit (return code, 4) .
389          |
390  format               | format :
391          |     control (7, format code, 0, return code) ;
392          |     IF return code = 1
393          |     THEN errorstop ("Formatieren unmoglich")
394          |     ELIF return code > 1
395          |     THEN errorstop ("Schreibfehler (Archiv)")
396          |     FI .
397          |
398          | ENDPROC format archive ;
399          |
400          | END PACKET basic archive;

```

```

1  archivesingle *****|PACKET archive single DEFINES                                (* Autor:
+                               |J.Liedtke*)                                                                (* Stand:
2
+                               |31.07.85 *)
3                               |archive ,
4                               |release ,
5                               |save ,
6                               |fetch ,
7                               |erase ,
8                               |check ,
9                               |exists ,
10                              |ALL ,
11                              |clear ,
12                              |list ,
13                              |format :
14
15
16
17                              |LET archive channel = 31 ,
18                              |main channel   = 1 ,
19
20                              |read error      = 92 ,
21
22                              |max files = 200 ,
23
24                              |start of volume = 1000 ,
25                              |end of volume = 1 ,
26                              |file header   = 3 ,
27
28                              |number of header blocks = 2 ,
29
30                              |quote          = "" ,
31                              |dummy name     = "_",
32                              |dummy date    = " " ,
33
34
35                              |HEADER = STRUCT (TEXT name, date, INT type, TEXT password) ;
36
37
38
39                              |TEXT VAR archive name := "" , write stamp ;
40
41                              |REAL VAR last access time := 0.0 ;
42
43                              |BOOL VAR was already write access := FALSE ;
44
45
46                              |DATASPACE VAR header space := nilspace , ds := nilspace ;
47                              |BOUND HEADER VAR header ;
48
49                              |TEXT VAR file name := "" ;
50
51                              |LET invalid   = 0 ,
52                              |read only    = 1 ,
53                              |valid       = 2 ;
54
55                              |LET accept read errors = TRUE ,
56                              |ignore read errors = FALSE ;
57
58
59                              |INT VAR directory state := invalid ;
60

```

```

61 |THESAURUS VAR directory , all names ;
62 |INT VAR dir index ;
63 |
64 |INT VAR archive size ;
65 |
66 |INT VAR end of volume block ;
67 |ROW max files INT VAR header block ;
68 |ROW max files TEXT VAR header date ;
69 |
70 |
71 |

72 archive .....|PROC archive (TEXT CONST name) :
73 |
74 |     disable stop ;
75 |     directory state := invalid ;
76 |     archive name := name ;
77 |     last access time := clock (1) .
78 |
79 |ENDPROC archive ;
80 |

81 release .....|PROC release (TASK CONST t) :
82 |
83 |     directory state := invalid
84 |
85 |ENDPROC release ;
86 |
87 |

88 accessarchive .....|PROC access archive :
89 |
90 |     IF directory state = invalid
91 |     THEN open archive
92 |     ELIF last access more than two seconds ago
93 |     THEN check volume name ;
94 |         new open if somebody changed medium
95 |     FI .
96 |

97 lastaccessmorethantwo |last access more than two seconds ago :
98 |     abs (clock (1) - last access time) > 2.0 .
99 |

100 newopenifsomebodychang |new open if somebody changed medium :
101 |     IF header.date <> write stamp
102 |     THEN directory state := invalid ;
103 |         access archive
104 |     FI .
105 |

106 openarchive |open archive :
107 |     directory state := invalid ;
108 |     check volume name ;
109 |     write stamp := header.date ;
110 |     was already write access := FALSE ;
111 |     read directory ;
112 |     make directory valid if no read errors occurred .
113 |

```

```

114 readdirectory      |read directory :
115                   |directory := empty thesaurus ;
116                   |rewind ;
117                   |get next header ;
118                   |WHILE header.type = file header REP
119                   |   IF directory CONTAINS header.name
120                   |     THEN rename (directory, header.name, dummy name)
121                   |   FI ;
122                   |insert (directory, header.name, dir index) ;
123                   |header block (dir index) := end of volume block ;
124                   |header date (dir index) := header.date ;
125                   |get next header ;
126                   |PER .
127
128 makedirectoryvalidifno |make directory valid if no read errors occurred :
129                   |IF directory state = invalid
130                   |  THEN directory state := valid
131                   |  FI .
132                   |
133                   |ENDPROC access archive ;
134
135 accessfile ..... |PROC access file (TEXT CONST name) :
136                   |
137                   |file name := name ;
138                   |dir index := link (directory, file name) .
139                   |
140                   |ENDPROC access file ;
141
142
143 checkvolumename ..... |PROC check volume name :
144                   |
145                   |disable stop ;
146                   |archive size := archive blocks ;
147                   |read volume header ;
148                   |IF header.type <> start of volume
149                   |  THEN simulate header (start of volume, "?????")
150                   |  ELIF header.name <> archive name
151                   |    THEN errorstop ("Archiv heisst "" + header.name + """)
152                   |  FI .
153
154 readvolumeheader    |read volume header :
155                   |rewind ;
156                   |read header ;
157                   |IF is error
158                   |  THEN clear error ;
159                   |    simulate header (start of volume, "?????")
160                   |  FI .
161                   |
162                   |ENDPROC check volume name ;
163
164 getnextheader ..... |PROC get next header :
165                   |
166                   |disable stop ;
167                   |skip dataspace ;

```

```

168      | IF NOT is error
169      |     THEN read header
170      |     FI ;
171      | IF is error
172      |     THEN clear error ;
173      |           directory state := read only ;
174      |           search header
175      |     FI ;
176      |     end of volume block := block number - number of header blocks .
177      |
178  searchheader | search header :
179      |     INT VAR ds pages ;
180      |     search dataspace (ds pages) ;
181      |     IF ds pages < 0
182      |         THEN simulate header (end of volume, "")
183      |     ELIF NOT is header space
184      |         THEN simulate header (file header, "?????" + text (block
+         number))
185      |     FI .
186      |
187  isheaderspace | is header space :
188      |     IF ds pages <> 1
189      |         THEN FALSE
190      |     ELSE remember position ;
191      |           read header ;
192      |           IF read error occurred
193      |               THEN clear error; back to old position; FALSE
194      |           ELIF header format looks ok
195      |               THEN TRUE
196      |           ELSE back to old position ; FALSE
197      |     FI
198      |     FI .
199      |
200  readerroroccurred | read error occurred :
201      |     is error CAND error code = read error .
202      |
203  headerformatlooksok | header format looks ok :
204      |     header.type = file header OR header.type = end of volume .
205      |
206  rememberposition | remember position :
207      |     INT CONST old block nr := block number .
208      |
209  backtooldposition | back to old position :
210      |     seek (old block nr) .
211      |
212      | ENDPROC get next header ;
213      |
214      |
215  fetch ..... | PROC fetch (TEXT CONST file name) :
216      |
217      |     fetch (file name, archive)
218      |
219      | ENDPROC fetch ;
220      |

```

```

221 fetch .....|PROC fetch (TEXT CONST file name, TASK CONST from) :
222
223 |   enable stop;
224 |   IF NOT (from = archive)
225 |     THEN errorstop ("Task gibt es nicht")
226 |   ELIF NOT exists (file name) COR overwrite permitted
227 |     THEN get archive file
228 |   FI .
229
230
231 |getarchivefile |get archive file:
232 |   last param (file name) ;
233 |   disable stop ;
234 |   continue (archive channel) ;
235 |   fetch file (file name) ;
236 |   last access time := clock (1) ;
237 |   continue (main channel) ;
238 |   IF NOT is error
239 |     THEN forget (file name, quiet) ;
240 |     copy (ds, file name)
241 |   FI ;
242 |   forget (ds) .
243
244 |overwritepermitted |overwrite permitted :
245 |   say ("eigene datei """);
246 |   say (file name) ;
247 |   yes ("" ueberschreiben") .
248
249 |ENDPROC fetch ;

```

```

250 fetchfile .....|PROC fetch file (TEXT CONST name) :
251
252 |   enable stop ;
253 |   access archive ;
254 |   access file (name) ;
255 |   IF no read error remarked
256 |     THEN disable stop ;
257 |     fetch ds (accept read errors) ;
258 |     IF read error occurred
259 |       THEN remark read error
260 |     FI ;
261 |     enable stop
262 |   ELSE fetch ds (ignore read errors)
263 |   FI .
264
265
266 |noreaderrorremarked |no read error remarked :
267 |   pos (name, " mit Lesefehler") = 0 .
268
269 |readerroroccurred |read error occurred :
270 |   is error AND error code = read error .
271
272 |remarkreaderror |remark read error :
273 |   dir index := link (directory, file name) ;
274 |   REP
275 |     file name CAT " mit Lesefehler" ;
276 |   UNTIL NOT (directory CONTAINS file name) PER ;

```

```

276          | IF LENGTH file name < 100
277          | THEN rename (directory, dir index, file name)
278          | FI .
279          |
280          |ENDPROC fetch file ;
281          |

282 fetchds .....|PROC fetch ds (BOOL CONST error accept) :
283          |
284          | enable stop ;
285          | IF file name <> dummy name
286          | THEN fetch from archive
287          | ELSE error ("Name unzulessig")
288          | FI .
289          |

290 fetchfromarchive |fetch from archive :
291          | IF file in directory
292          | THEN position to file ;
293          |         forget (ds) ;
294          |         ds := nilspace ;
295          |         read (ds, 30000, error accept) ;
296          | ELIF directory state = read only
297          | THEN error ("gibt es nicht (oder Lesefehler)")
298          | ELSE error ("gibt es nicht")
299          | FI .
300          |

301 positiontofile |position to file :
302          | seek (header block (dir index) + number of header blocks) .
303          |

304 fileindirectory |file in directory : dir index > 0 .
305          |
306          |ENDPROC fetch ds ;
307          |

308 erase .....|PROC erase :
309          |
310          | erase (last param)
311          |
312          |ENDPROC erase ;
313          |

314 erase .....|PROC erase (TEXT CONST file name) :
315          |
316          | erase (file name, archive)
317          |
318          |ENDPROC erase ;
319          |

320 erase .....|PROC erase (TEXT CONST file name, TASK CONST dest) :
321          |
322          | IF dest = archive
323          | THEN disable stop ;
324          |         continue (archive channel) ;
325          |         erase on archive (file name) ;
326          |         last access time := clock (1) ;

```

```

327 |           continue (main channel)
328 |           ELSE errorstop ("Task gibt es nicht")
329 |       FI
330 |
331 |   ENDPROC erase ;
332 |
333 | eraseonarchive .....|PROC erase on archive (TEXT CONST file name) :
334 |
335 |       enable stop ;
336 |       access archive ;
337 |       access file (file name) ;
338 |       continue (main channel) ;
339 |       IF NOT file in directory
340 |           THEN putline ("gibt es nicht") ;
341 |               LEAVE erase on archive
342 |       ELIF NOT yes (""+file name+"" loeschen")
343 |           THEN LEAVE erase on archive
344 |       FI ;
345 |       continue (archive channel) ;
346 |       erase archive entry .
347 |
348 | fileindirectory      |file in directory : dir index > 0 .
349 |
350 |   ENDPROC erase on archive ;
351 |
352 | erasearchiveentry .....|PROC erase archive entry :
353 |
354 |       IF directory state = read only
355 |           THEN errorstop ("save/'erase' wegen Lesefehler verboten")
356 |           ELSE update write stamp if first write access ;
357 |               erase archive
358 |       FI .
359 |
360 | updatewritestampiffirs|update write stamp if first write access :
361 |       IF NOT was already write access
362 |           THEN rewind ;
363 |               write stamp := text (clock (1), 13, 1) ;
364 |               write header (archive name, write stamp, start of volume) ;
365 |               was already write access := TRUE
366 |       FI .
367 |
368 | erasearchive         |erase archive :
369 |       IF file in directory
370 |           THEN IF is last file of archive
371 |               THEN cut off all erased files
372 |               ELSE rename to dummy
373 |           FI
374 |       FI .
375 |
376 | fileindirectory      |file in directory :           dir index > 0 .
377 |

```

```

378 islastfileofarchive | is last file of archive : dir index = highest entry (directory) .
379
380 cutoffallerasedfiles | cut off all erased files :
381 | directory state := invalid ;
382 | REP
383 | delete (directory, dir index) ;
384 | dir index DECR 1
385 | UNTIL dir index = 0 COR name (directory, dir index) <> dummy name
+ | PER ;
386 | behind last valid file ;
387 | write end of volume ;
388 | directory state := valid .
389
390 behindlastvalidfile | behind last valid file :
391 | seek (header block (dir index + 1)) ;
392 | end of volume block := block number .
393
394 renametodummy | rename to dummy :
395 | directory state := invalid ;
396 | to file header ;
397 | read header ;
398 | to file header ;
399 | header.name := dummy name ;
400 | header.date := dummy date ;
401 | write (header space) ;
402 | rename (directory, file name, dummy name) ;
403 | header date (dir index) := dummy date ;
404 | directory state := valid .
405
406 tofileheader | to file header :
407 | seek (header block (dir index)) .
408
409 |ENDPROC erase archive entry ;
410
411 save ..... |PROC save :
412 |
413 | save (last param)
414 |
415 |ENDPROC save ;
416
417 save ..... |PROC save (TEXT CONST file name) :
418 |
419 | save (file name, archive)
420 |
421 |ENDPROC save ;
422
423 save ..... |PROC save (TEXT CONST file name, TASK CONST to) :
424 |
425 | IF to = archive
426 | THEN disable stop ;
427 | continue (archive channel) ;
428 | save to archive (file name) ;

```

```

429                                     last access time := clock (1) ;
430                                     continue (main channel)
431                                     ELSE errorstop ("Task gibt es nicht")
432                                     FI .
433
434                                     ENDPROC save ;
435
436 savetoarchive ..... PROC save to archive (TEXT CONST file name) :
437
438                                     enable stop ;
439                                     access archive ;
440                                     access file (file name) ;
441                                     continue (main channel) ;
442                                     IF file in directory
443                                     THEN IF NOT yes (""+file name+"" ueberschreiben")
444                                     THEN LEAVE save to archive
445                                     FI
446                                     FI ;
447                                     continue (archive channel) ;
448                                     access archive ;
449                                     access file (file name) ;
450                                     erase archive entry ;
451                                     IF file name = dummy name
452                                     THEN error ("Name unzulässig")
453                                     ELIF file too large OR highest entry (directory) >= max files
454                                     THEN error ("kann nicht geschrieben werden (Archiv voll)")
455                                     ELSE write new file
456                                     FI .
457
458 fileindirectory file in directory : dir index > 0 .
459
460 filetoolarge file too large :
461 end of volume block + ds pages (ds) + 5 > archive size .
462
463 writenewfile write new file :
464 seek (end of volume block) ;
465 disable stop ;
466 write file (file name, old (file name)) ;
467 IF is error
468 THEN seek (end of volume block)
469 ELSE insert (directory, file name, dir index) ;
470 remember begin of header block ;
471 remember date
472 FI ;
473 write end of volume .
474
475 rememberbeginofheaderb remember begin of header block :
476 header block (dir index) := end of volume block .
477
478 rememberdate remember date :
479 header date (dir index) := date .
480
481                                     ENDPROC save to archive ;
482

```

```

483 writefile .....|PROC write file (TEXT CONST file name, DATASPACE CONST ds) :
484
485 |       enable stop ;
486 |       write header (file name, date, file header) ;
487 |       write (ds)
488 |
489 |ENDPROC write file ;
490

491 writeendofvolume .....|PROC write end of volume :
492
493 |       disable stop ;
494 |       end of volume block := block number ;
495 |       write header ("", "", end of volume)
496 |
497 |ENDPROC write end of volume ;
498

499 writeheader .....|PROC write header (TEXT CONST name, date, INT CONST header type) :
500
501 |       forget (header space) ;
502 |       header space := nilspace ;
503 |       header := header space ;
504 |
505 |       header.name := subtext (name,1,100) ;
506 |       header.date := date ;
507 |       header.type := header type ;
508 |
509 |       write (header space)
510 |
511 |ENDPROC write header ;
512

513 readheader .....|PROC read header :
514
515 |       forget (header space) ;
516 |       header space := nilspace ;
517 |       read (header space, 1, accept read errors) ;
518 |       header := header space .
519 |
520 |ENDPROC read header ;
521

522 simulateheader .....|PROC simulate header (INT CONST type, TEXT CONST name) :
523
524 |       forget (header space) ;
525 |       header space := nilspace ;
526 |       header := header space ;
527 |       header.name := name ;
528 |       header.date := "???.???.??";
529 |       header.type := type ;
530 |       header.password := ""
531 |
532 |ENDPROC simulate header ;
533

```

```

534 check .....|PROC check (TEXT CONST name, TASK CONST from) :
535
536           |   IF from = archive
537           |       THEN check file
538           |       ELSE errorstop ("Task gibt es nicht")
539           |   FI .
540
541 checkfile   |check file :
542           |   access archive ;
543           |   access file (name) ;
544           |   IF file in directory
545           |       THEN position to file ;
546           |           disable stop ;
547           |           check read ;
548           |           IF is error
549           |               THEN clear error; error ("fehlerhaft")
550           |               ELSE last access time := clock (1) ;
551           |                   putline ("" + file name + "" ohne Fehler gelesen)
552           |           FI
553           |       ELSE error ("gibt es nicht")
554           |   FI .
555
556 fileindirectory |file in directory : dir index > 0 .
557
558 positiontofile |position to file :
559           |   seek (header block (dir index) + number of header blocks) .
560
561
562           |ENDPROC check ;
563
563 exists .....|BOOL PROC exists (TEXT CONST name, TASK CONST from) :
564
565           |   IF from = archive
566           |       THEN access archive ;
567           |           access file (name) ;
568           |           file in directory
569           |       ELSE FALSE
570           |   FI .
571
572 fileindirectory |file in directory : dir index > 0 .
573
574
575           |ENDPROC exists ;
576
576 list .....|PROC list (TASK CONST from) :
577
578           |   forget (ds) ;
579           |   ds := nilspace ;
580           |   FILE VAR list file := sequential file (output, ds) ;
581           |   list (list file, from) ;
582           |   modify (list file) ;
583           |   show (list file) ;
584           |   forget (ds) .
585
586           |ENDPROC list ;
587

```

```

588 list .....|PROC list (FILE VAR list file, TASK CONST from) :
589
590           |   IF from = archive
591           |       THEN disable stop ;
592           |           continue (archive channel) ;
593           |           list archive (list file) ;
594           |           last access time := clock (1) ;
595           |           continue (main channel)
596           |       ELIF from = myself                      (* R. Nolting
+           |           25.10.84 *)
597           |       THEN list(listfile)
598           |       ELSE errorstop ("Task gibt es nicht")
599           |       FI .
600
601           |ENDPROC list ;
602
603 listarchive .....|PROC list archive (FILE VAR list file) :
604
605           |   enable stop ;
606           |   access archive ;
607           |   open list file ;
608           |   INT VAR file number := 0 ;
609           |   get (directory, file name, file number) ;
610           |   WHILE file number > 0 REP
611           |       generate list line ;
612           |       get (directory, file name, file number)
613           |   PER ;
614           |   IF directory state = read only
615           |       THEN putline (list file, "Lesefehler: Evtl. fehlen Einträge")
616           |   FI ;
617           |   write list head .
618
619 openlistfile      |open list file :
620                   |   output (list file) ;
621                   |   putline (list file, "") .
622
623 generatelistline |generate list line :
624                   |   write (list file, header date (file number)) ;
625                   |   write (list file, text (file blocks DIV 2, 5)) ;
626                   |   write (list file, " K ") ;
627                   |   IF header.name = dummy name
628                   |       THEN write (list file, dummy name)
629                   |       ELSE write (list file, quote) ;
630                   |           write (list file, file name ) ;
631                   |           write (list file, quote)
632                   |   FI ;
633                   |   line (list file) .
634
635 fileblocks        |file blocks :
636                   |   IF file number < highest entry (directory)
637                   |       THEN header block (file number+1) - header block (file number)
638                   |       ELSE end of volume block - header block (file number)
639                   |   FI .
640

```

```

641 writelisthead |write list head :
642 |headline (list file, archive name +
643 |      " (" + used + " K belegt von " + text (archive size DIV
644 |      2) + " K")" ) .
645 used |used : text ((end of volume block + 3) DIV 2) .
646 |
647 |ENDPROC list archive ;
648 |

```

```

649 ALL .....|THESAURUS OP ALL (TASK CONST from) :
650 |
651 |IF from = myself
652 |  THEN all
653 |  ELIF from = archive
654 |    THEN disable stop ;
655 |      continue (archive channel) ;
656 |      get all from archive ;
657 |      last access time := clock (1) ;
658 |      continue (main channel) ;
659 |      enable stop ;
660 |      all names
661 |  ELSE errorstop ("Task gibt es nicht") ;
662 |      empty thesaurus
663 |  FI .
664 |
665 |ENDOP ALL ;
666 |

```

```

667 getallfromarchive .....|PROC get all from archive :
668 |
669 |enable stop ;
670 |access archive ;
671 |all names := directory ;
672 |WHILE all names CONTAINS dummy name REP
673 |  delete (all names, dummy name, dir index)
674 |  PER .
675 |
676 |ENDPROC get all from archive ;
677 |

```

```

678 clear .....|PROC clear (TASK CONST dest) :
679 |
680 |IF dest = archive
681 |  THEN disable stop ;
682 |      continue (archive channel) ;
683 |      clear archive ;
684 |      continue (main channel)
685 |  ELSE errorstop ("Task gibt es nicht")
686 |  FI .
687 |
688 |ENDPROC clear ;
689 |

```

```

690 cleararchive .....|PROC clear archive :
691 |
692 |archive size := archive blocks ;

```

```

693      ask for erase all ;
694      directory state := invalid ;
695      rewind ;
696      write header (archive name, text (clock(1),13,1), start of volume)
697      write end of volume .
698
699      askforeraseall      ask for erase all :
700      rewind ;
701      disable stop ;
702      read header ;
703      IF is error OR
704      LENGTH header.name < @ OR LENGTH header.name > 100 OR is error
705      THEN header.name := "" ;
706      clear error
707      FI ;
708      enable stop ;
709      continue (main channel) ;
710      IF header.name <> ""
711      THEN IF NOT yes ("archiv ""+header.name+"" loeschen")
712      THEN LEAVE clear archive
713      FI
714      ELSE IF NOT yes ("archiv initialisieren")
715      THEN LEAVE clear archive
716      FI
717      FI ;
718      continue (archive channel) .
719
720      ENDPROC clear archive ;
721

```

```

722      format ..... PROC format (INT CONST format code, TASK CONST dest) :
723
724      IF dest = archive
725      THEN IF yes ("?"Formatieren ueberschreibt alles! Richtige
726      +      Diskette eingelegt")
727      THEN disable stop ;
728      continue (archive channel) ;
729      format archive (format code) ;
730      directory state := invalid ;
731      rewind ;
732      write header ( archive name, text (clock (1), 13, 1)
733      +      ,start of volume) ;
734      write end of volume ;
735      continue (main channel)
736      FI
737      ELSE errorstop ("Task gibt es nicht")
738      FI .
739      ENDPROC format ;

```

```

740      format ..... PROC format (TASK CONST dest) :
741
742      format (@, dest)
743
744      ENDPROC format ;
745

```

Zeile **** E L A N EUMEL 1.8 **** 10.11.86 **** archive single

```
746 error .....|PROC error (TEXT CONST error msg) :  
747             |  
748             | errorstop ("" + file name + "" + error msg)  
749             |  
750             |ENDPROC error ;  
751             |  
752             |ENDPACKET archive single ;
```

```

1      |(* ----- VERSION 4      22.04.86 ----- *)
2  konfigurieren *****|PACKET konfigurieren DEFINES      (* Autor: D.Heinrichs *)
3
4
5
6      |      ansi cursor,
7      |      baudrate ,
8      |      bits ,
9      |      cursor logic ,
10     |      elbit cursor ,
11     |      enter incode ,
12     |      enter outcode ,
13     |      flow ,
14     |      input buffer size ,
15     |      link ,
16     |      new configuration ,
17     |      new type ,
18     |      ysize :
19
20     |LET max dtype nr = 5, (* maximum number of active device tables *)
21     |device table = 32000,
22     |ack = 0 ;
23
24
25     |INT VAR next outstring,
26     |      next instring;
27
28     |BOUND STRUCT (ALIGN space,      (*
29     |      umsetzcodetabelle *)
30     |      ROW 128 INT outcodes,
31     |      ROW 64 INT outstrings,
32     |      ROW 64 INT instrings) VAR x;
33
34     |ROW max dtype nr DATASPACE VAR device code table;
35
36     |THESAURUS VAR dtypes ;
37
38
39  newconfiguration .....|PROC new configuration :
40
41     |      dtypes := empty thesaurus ;
42     |      INT VAR i ;
43     |      insert (dtypes, "psi", i) ;
44     |      insert (dtypes, "transparent", i) ;
45     |      FOR i FROM 1 UPTO max dtype nr REP
46     |        forget (device code table (i))
47     |      PER .
48
49     |ENDPROC new configuration ;
50
51
52  blockout .....|PROC block out (DATASPACE CONST ds, INT CONST page, code):
53     |      INT VAR err;
54     |      block out (ds,page,0,code,err);
55     |      announce error (err)
56     |END PROC block out;
57

```

```

58 announceerror .....|PROC announce error (INT CONST err):
59                       |SELECT err OF
60                       |CASE 0:
61                       |CASE 1: errorstop ("unbekanntes Terminalkommando")
62                       |CASE 2: errorstop ("Nummer der Terminal-Typ-Tabelle falsch")
63                       |CASE 3: errorstop ("falsche Terminalnummer")
64                       |OTHERWISE errorstop ("blockout: unzulessiger Kanal")
65                       |ENDSELECT
66                       |END PROC announce error;
67                       |

68 flow .....|PROC flow (INT CONST nr, INT CONST dtype):
69           |control (6, dtype, nr)
70           |END PROC flow;
71           |

72 ysize .....|PROC ysize (INT CONST channel ,new size, INT VAR old size) :
73           |control (11, channel, new size, old size)
74           |ENDPROC ysize ;
75           |

76 inputbuffersize .....|PROC input buffer size (INT CONST nr,size):
77           |INT VAR err;
78           |control (2,nr,size,err)
79           |END PROC input buffer size;
80           |

81 baudrate .....|PROC baudrate (INT CONST nr, rate) :
82           |control (8, rate, nr)
83           |ENDPROC baudrate ;
84           |

85 bits .....|PROC bits (INT CONST channel, number, parity) :
86           |bits (channel, number-1 + 8*parity)
87           |ENDPROC bits ;
88           |

89 bits .....|PROC bits (INT CONST channel, key) :
90           |control (9, key, channel)
91           |ENDPROC bits ;
92           |

93 control .....|PROC control (INT CONST function, key, channel) :
94           |
95           |INT VAR err ;
96           |IF key > -128 AND key < 127
97           |THEN control (function, channel, key, err)
98           |ELIF key = -128
99           |THEN control (function, channel, -maxint-1, err)
100          |
101          |FI
102          |ENDPROC control ;
103          |
104          |

```

```

105 newtype .....|PROC new type (TEXT CONST dtype):
106                | x := new (dtype);
107                | type (old (dtype), device table);
108                | next outstring := 4;
109                | next instring := 0;
110                | INT VAR i;
111                | (* Defaults, damit trmpret den cursor mitfuehrt: *)
112                |   FOR i FROM 1 UPTO 6 REP
113                |     enter outcode (i,i)
114                |     PER;
115                |     enter outcode (8,8);
116                |     enter outcode (10,10);
117                |     enter outcode (13,13);
118                |     enter outcode (14,126);
119                |     enter outcode (15,126);
120                | END PROC new type;
121                |
122
122 activatedtype .....|INT PROC activate dtype (TEXT CONST dtype):
123                |
124                | INT VAR i := link (dtypes, dtype);
125                | IF (exists (dtype) CAND type (old (dtype)) = device table)
126                |   THEN IF i <= 0
127                |     THEN insert (dtypes, dtype, i);
128                |     FI;
129                |     forget(device code table (i-2));
130                |     device code table (i-2) := old (dtype)
131                |   FI;
132                | IF i > max dtype nr +2 (* 5 neue Typen erlaubt *)
133                |   THEN delete (dtypes,i);
134                |     error stop ("Anzahl Terminaltypen > "+text (i));0
135                |   ELIF i <= 0
136                |     THEN error stop ("Unbekannter Terminaltyp" + dtype); 0
137                |   ELSE i
138                |   FI.
139                |
140                | END PROC activate dtype;
141                |
142
142 link .....|PROC link (INT CONST nr, TEXT CONST dtype):
143                |
144                | INT VAR lst nr := activate dtype (dtype)-3;
145                | IF lst nr < 0
146                |   THEN lst nr INCR 256 (* fuer std terminal und std device *)
147                |   ELSE blackout (device code table(lst nr+1), 2, lst nr);
148                |   FI;
149                | INT VAR err := 0;
150                | control (1,nr,lst nr,err) ;
151                | announce error(err)
152                |
153                | END PROC link;
154                |
155                |
156
156 enteroutcode .....|PROC enter outcode (INT CONST eumel code, ziel code):
157                |
158                | IF ziel code < 128
159                |   THEN simple entry (eumel code, ziel code)
160                |   ELSE enter outcode (eumel code, 0, code (ziel code))

```

```

161          |   FI .
162          |
163          |ENDPROC enter outcode ;
164          |

165 simpleentry .....|PROC simple entry (INT CONST eumel code, ziel code) :
166          |
167          |   INT CONST position := eumel code DIV 2 +1,
168          |   teil := eumel code - 2*position + 2;
169          |   TEXT VAR h :="      ";
170          |   replace (h,1,out word);
171          |   replace (h,1+teil,code (ziel code));
172          |   out word := (h ISUB 1).
173          |
174          |   outword                out word: x.outcodes (position).
175          |
176          |END PROC simple entry ;
177          |

178 enteroutcode .....|PROC enter outcode (INT CONST eumel code, wartezeit,
179          |               TEXT CONST sequenz):
180          |
181          |   INT VAR i;
182          |   simple entry (eumel code, next outstring + 128);
183          |   enter part (x.outstrings, next outstring, wartezeit);
184          |   FOR i FROM 1 UPTO length (sequenz) REP
185          |       enter part (x.outstrings, next outstring + i, code
186          |           (sequenzSUB1))
187          |   PER;
188          |   next outstring INCR length (sequenz)+2;
189          |   abschluss.
190          |
191          |   abschluss:
192          |       enter part (x.outstrings, next outstring-1, 0)
193          |END PROC enter outcode;

194 enteroutcode .....|PROC enter outcode (INT CONST eumelcode, TEXT CONST wert):
195          |   enter outcode (eumelcode,code(wert))
196          |END PROC enter outcode;
197          |

198 enterpart .....   |PROC enter part (ROW 64 INT VAR a,INT CONST index, wert):
199          |   INT CONST position := index DIV 2 +1,
200          |   teil := index - 2*position + 2;
201          |   IF position > 64 THEN errorstop ("Ueberlauf der
202          |       Terminaltypabelle") FI;
203          |   TEXT VAR h :="      ";
204          |   replace (h,1,out word);
205          |   replace (h,1+teil,code (wert));
206          |   out word := (h ISUB 1).
207          |
208          |   outword                out word: a (position).
209          |END PROC enter part;

```

210

```

211 enterincode .....|PROC enter incode (INT CONST elan code, TEXT CONST sequenz):
212                   |IF elan code > 254 OR elan code < 0 THEN errorstop ("kein
+                   |   Eingabecode")
213                   |ELSE
214                   |   INT VAR i;
215                   |   enter part (x.instrings, next instring, elan code);
216                   |   FOR i FROM 1 UPTO length (sequenz) REP
217                   |     enter part (x.instrings, next instring + i, code
+                   |       (sequenzSUBi))
218                   |   PER;
219                   |   next instring INCR length (sequenz)+2;
220
221                   |   FI
222
223                   |END PROC enter incode;
224

```

```

225 cursorlogic .....|PROC cursor logic (INT CONST dist, TEXT CONST pre, mid, post):
226
227                   |   cursor logic (dist,255,pre,mid,post)
228
229                   |END PROC cursor logic;
230

```

```

231 ansicursor .....|PROC ansi cursor (TEXT CONST pre, mid, post):
232
233                   |   cursor logic (0, 1, pre, mid, post)
234
235                   |END PROC ansi cursor;
236

```

```

237 cursorlogic .....|PROC cursor logic (INT CONST dist, modus, TEXT CONST pre, mid, post)
238
239                   |   enter part (x.outstrings,2,dist);
240                   |   enter part (x.outstrings,3,dist);
241                   |   enter part (x.outstrings,0,modus);
242                   |   enter part (x.outstrings,1,modus);
243                   |   enter outcode (6,0,pre+"0"y"+mid+"0"x"+post+"0")
244
245                   |END PROC cursor logic;
246

```

```

247 elbitcursor .....|PROC elbit cursor:
248                   |   cursor logic (0, "27", "", "");
249                   |   enter part (x.outstrings,0,2);
250                   |   enter part (x.outstrings,1,255);
251                   |END PROC elbit cursor;
252
253                   |ENDPACKET konfigurieren;

```

```

1      (* ----- VERSION 11      10.06.86 ----- *)
2  configuratorsingle ***** PACKET configurator single DEFINES
3
4      configure ,
5      exec configuration ,
6      setup :
7
8      LET baudrates      = "1"50"2"75"3"110"4"134.5"5"150"6"300"7"600
9      "8"1200"9"1800"10"2400"11"3600"12"4800"13"7200
10     "14"9600"15"19200"16"38400"17" ,
11     parities           = "0"no"1"odd"2"even"3" ,
12     bits per char      = "0"1"1"2"2"3"3"4"4"5"5"6"6"7"7"8"8" ,
13     stopbits           = "0"1"1"1.5"2"2"3" ,
14     flow modes         = "0"ohne Protokoll"1"XON/XOFF"2"RTS/CTS
15     "3"4"5"XON/XOFF - ausgabeseitig"6"RTS/CTS - ausgabeseitig"7"8"
16     "9"XON/XOFF - eingabeseitig"10"RTS/CTS - eingabeseitig"11" ,
17
18     ok                  = "j" ,
19     esc                  = "27" ,
20     cr                   = "13" ,
21     right                = "2" ,
22
23     psi                  = "psi" ,
24     transparent          = "transparent" ,
25
26     std rate             = 14 ,
27     std bits             = 22 ,
28     std flow             = 0 ,
29     std inbuffer size   = 16 ,
30
31     device table        = 32000 ,
32
33     max edit terminal    = 15 ,
34     configuration channel = 32 ,
35
36     CONF = STRUCT (TEXT dev type,
37                 INT baud, bits par stop, flow control, inbuffer
38                 size) ;
39
40     BOUND ROW max edit terminal CONF VAR conf ;
41
42     INT VAR channel no ;
43
44     TEXT VAR prelude , last feature , answer ;
45
46
47
48  shardpermits ..... BOOL PROC shard permits (INT CONST code, key) :
49
50     INT VAR reply ;
51     IF key > -128
52     THEN control (code, channel no, key, reply)
53     ELSE control (code, channel no, -maxint-1, reply)
54     FI ;
55     reply = 0 .
56
57     ENDPROC shard permits ;
58

```

```

59 askuser ..... PROC ask user (TEXT CONST feature, question) :
60
61 | last feature := feature ;
62 | put question ;
63 | skip pretyped chars ;
64 | get valid answer .
65
66 putquestion | put question :
67 | clear line ;
68 | out (prelude) ;
69 | out (feature) ;
70 | out (question) ;
71 | out (" (j/n) ") .
72
73 clearline | clear line :
74 | out (cr) ;
75 | 79 TIMESOUT " " ;
76 | out (cr) .
77
78 skippretypedchars | skip pretyped chars :
79 | REP UNTIL incharety = "" PER .
80
81 getvalidanswer | get valid answer :
82 | REP
83 | inchar (answer)
84 | UNTIL pos ("jJyYn"27", answer) > 0 PER ;
85 | IF answer > ""31""
86 | THEN out (answer)
87 | FI ;
88 | out (cr) ;
89 | normalize answer .
90
91 normalizeanswer | normalize answer :
92 | IF pos ("jJyY", answer) > 0
93 | THEN answer := ok
94 | FI .
95
96 | ENDPROC ask user ;
97
98 yes ..... BOOL PROC yes (TEXT CONST question) :
99
100 | ask user ("", question) ;
101 | answer = ok
102
103 | ENDPROC yes ;
104
105 chosekey ..... PROC chose key (INT VAR old key, INT CONST max key, TEXT CONST key
+ | string,
106 | key entity, BOOL PROC (INT CONST) shard permits):
107
108 | IF shard permits at least one standard key
109 | THEN try all keys
110 | FI .

```

```

111
112   shardpermitsatleastone | shard permits at least one standard key :
113                           |   INT VAR key ;
114                           |   FOR key FROM 0 UPTO max key REP
115                           |     IF shard permits (key)
116                           |       THEN LEAVE shard permits at least one standard key WITH TRUE
117                           |     FI
118                           |   PER ;
119                           |   FALSE .
120
121   tryallkeys               | try all keys :
122                           |   key := old key ;
123                           |   REP
124                           |     examine this key ;
125                           |     next key
126                           |   PER .
127
128   examinethiskey          | examine this key :
129                           |   IF shard permits (key) CAND key value <> ""
130                           |     THEN ask user (key value, key entity) ;
131                           |     IF answer = ok
132                           |       THEN chose this key
133                           |     ELIF answer = esc
134                           |       THEN key := -129
135                           |     FI
136                           |   FI .
137
138   keyvalue                | key value :
139                           |   IF key >= 0
140                           |     THEN subtext (key string, key pos + 1, next key pos - 1)
141                           |     ELSE text (key)
142                           |   FI .
143
144   keypos                   | key pos      : pos (key string, code (key)) .
145   nextkeypos               | next key pos : pos (key string, code (key+1)) .
146
147   chosethiskey            | chose this key :
148                           |   remember calibration ;
149                           |   old key := key ;
150                           |   LEAVE chose key .
151
152   nextkey                  | next key :
153                           |   IF key < max key
154                           |     THEN key INCR 1
155                           |     ELSE key := 0
156                           |   FI .
157
158   remembercalibration     | remember calibration :
159                           |   prelude CAT last feature ;
160                           |   prelude CAT ", " .
161
162                           | ENDPROC chose key ;
163

```

```

164 rateok .....|BOOL PROC rate ok (INT CONST key) :
165                |
166                |   shard permits (8, key)
167                |
168                |ENDPROC rate ok ;
169                |

170 bitsok .....|BOOL PROC bits ok (INT CONST key) :
171                |
172                |   IF key < 0
173                |     THEN shard permits (9, key)
174                |     ELSE some standard combination ok
175                |   FI .
176                |

177 somestandardcombinatio|some standard combination ok :
178                |   INT VAR combined := key ;
179                |   REP
180                |     IF shard permits (9, combined)
181                |       THEN LEAVE bits ok WITH TRUE
182                |     FI ;
183                |     combined INCR 8
184                |   UNTIL combined > 127 PER ;
185                |   FALSE
186                |
187                |ENDPROC bits ok ;
188                |

189 parityok .....|BOOL PROC parity ok (INT CONST key) :
190                |
191                |   INT VAR combined := 8 * key + data bits ;
192                |   key >= 0 AND (shard permits (9, combined)      OR
193                |     shard permits (9, combined + 32) OR
194                |     shard permits (9, combined + 64) )
195                |
196                |ENDPROC parity ok ;
197                |

198 stopbitsok .....|BOOL PROC stopbits ok (INT CONST key) :
199                |
200                |   key >= 0 AND shard permits (9, 32 * key + 8 * parity + data bits)
201                |
202                |ENDPROC stopbits ok ;
203                |

204 flowmodeok .....|BOOL PROC flow mode ok (INT CONST key) :
205                |
206                |   shard permits (6, key)
207                |
208                |ENDPROC flow mode ok ;
209                |
210                |
211                |
212                |   INT VAR operators channel ,
213                |     data bits ,
214                |     parity ,
215                |     stop ;
216                |

```

```

217 |TEXT VAR table name, dummy ;
218 |
219 |
220 |configure .....|PROC configurate :
221 |
222 |   new configuration ;
223 |   access configuration table ;
224 |   show all device types ;
225 |   channel no := 1 ;
226 |   REP
227 |     IF channel hardware exists
228 |       THEN try this channel ;
229 |         setup this channel
230 |     FI ;
231 |     channel no INCR 1
232 | UNTIL channel no > 15 PER ;
233 | prelude := "" ;
234 | IF yes ("Koennen unbenutzte Geraetetypen geloescht werden")
235 | THEN forget unused device tables
236 | FI .
237 |
238 |accessconfigurationtab|access configuration table :
239 |   IF exists ("configuration")
240 |     THEN conf := old ("configuration")
241 |     ELSE conf := new ("configuration") ;
242 |     initialize configuration
243 |   FI .
244 |
245 |initializeconfiguratio|initialize configuration :
246 |   FOR channel no FROM 1 UPTO max edit terminal REP
247 |     conf (channel no) :=
248 |       CONF:(transparent, std rate, std bits, std flow, std inbuffer
249 |         +
250 |         size)
251 |     PER ;
252 |     conf (1).dev type := psi .
253 |
254 |showalldvicetypes|show all device types :
255 |   show prelude ;
256 |   begin list ;
257 |   get list entry (table name, dummy) ;
258 |   WHILE table name <> "" REP
259 |     IF dataspace is device table
260 |       THEN show table name
261 |     FI ;
262 |     get list entry (table name, dummy)
263 |   PER ;
264 |   line (2) .
265 |
266 |showprelude|show prelude :
267 |   line (30) ;
268 |   outtext (psi, 1, 20) ;
269 |   outtext (transparent, 1, 20) .

```

```

269      dataspaceisdevicetable |dataspace is device table :
270      |                          type (old (table name)) = device table .
271      |
272      showtablename           |show table name :
273      |                          outtext (table name, 1, 20) .
274      |
275      trythischannel          |try this channel :
276      |                          prelude := "Kanal " ;
277      |                          ask user ("", text (channel no)) ;
278      |                          IF answer = ok
279      |                              THEN prelude CAT text (channel no) + ": " ;
280      |                              get configuration from user (conf (channel no)) ;
281      |                              line
282      |                          FI .
283      |
284      channelhardwareexists   |channel hardware exists :
285      |                          operators channel := channel ;
286      |                          INT VAR channel type ;
287      |                          disable stop ;
288      |                          continue (channel no) ;
289      |                          IF is error
290      |                              THEN IF error message = "kein Kanal"
291      |                                  THEN channel type := 0
292      |                                  ELSE channel type := inout mask
293      |                              FI
294      |                              ELSE get channel type from shard
295      |                          FI ;
296      |                          clear error ;
297      |                          disable stop ;
298      |                          continue operators channel ;
299      |                          (channel type AND inout mask) <> 0 .
300      |
301      getchanneltypefromshar |get channel type from shard :
302      |                          control (1, 0, 0, channel type) .
303      |
304      inoutmask                |inout mask : 3 .
305      |
306      forgetunuseddevicetabl |forget unused device tables :
307      |                          begin list ;
308      |                          get list entry (table name, dummy) ;
309      |                          WHILE table name <> "" REP
310      |                              IF type (old (table name)) = device table
311      |                                  THEN forget if unused
312      |                              FI ;
313      |                              get list entry (table name, dummy)
314      |                          PER .
315      |
316      forgetifunused          |forget if unused :
317      |                          FOR channel no FROM 1 UPTO max edit terminal REP
318      |                              IF conf (channel no).dev type = table name
319      |                                  THEN LEAVE forget if unused
320      |                              FI
321      |                          PER ;
322      |                          forget (table name, quiet) .
323      |

```

```

324      setupthischannel      |setup this channel :
325                          | operators channel := channel ;
326                          | disable stop ;
327                          | continue (configuration channel) ;
328                          | set up channel (channel no, conf (channel no)) ;
329                          | continue operators channel .
330
331      continueoperatorschann |continue operators channel :
332                          | continue (operators channel) ;
333                          | IF is error
334                          |     THEN clear error ;
335                          |          LEAVE configurate
336                          | FI ;
337                          | enable stop .
338
339                          |ENDPROC configurate ;
340
341  getconfigurationfromus ... |PROC get configuration from user (CONF VAR conf) :
342
343                          | get device type ;
344                          | get baud rate ;
345                          | get bits and parity and stopbits ;
346                          | get protocol ;
347                          | get buffer size .
348
349
350  getdevicetype             |get device type :
351                          | begin list ;
352                          | table name := conf.dev type ;
353                          | IF NOT is valid device type
354                          |     THEN next device type
355                          | FI ;
356                          | REP
357                          |     IF NOT (table name = transparent AND channel no = 1)
358                          |         THEN ask user ("", table name) ;
359                          |             IF answer = ok COR was esc followed by type table name
360                          |                 THEN IF is valid device type
361                          |                     THEN remember device type ;
362                          |                         LEAVE get device type
363                          |                     ELSE out ("?" unbekannter Typ"); pause (20)
364                          |             FI
365                          |             FI
366                          |             next device type
367                          | PER .
368
369
370  wasescfollowedbytypeta    |was esc followed by type table name :
371                          | IF answer = esc
372                          |     THEN 9 TIMESOUT right ;
373                          |         put ("Typ:");
374                          |         editget (table name) ;
375                          |         TRUE
376                          |     ELSE FALSE
377                          | FI .
378

```

```

379   isvaliddevicetype | is valid device type :
380   | table name = psi OR table name = transparent OR
381   | (exists (table name) CAND type (old (table name)) = device table)
382
383   rememberdevicetype | remember device type :
384   | prelude CAT table name ;
385   | conf.dev type := table name ;
386   | prelude CAT ", " .
387
388   nextdevicetype     | next device type :
389   | IF table name = psi
390   | THEN table name := transparent
391   | ELSE IF table name = transparent
392   | THEN begin list
393   |     FI ;
394   |     search next device type space
395   | FI .
396
397   searchnextdevicetypesp | search next device type space :
398   | REP
399   |     get list entry (table name, dummy)
400   | UNTIL table name = "" COR type (old (table name)) = device table
401   |     PER;
402   | IF table name = ""
403   | THEN table name := psi
404   | FI .
405
406   getbaudrate        | get baudrate :
407   | chose key (conf.baud, 16, baudrates, " Baud", PROC rate ok) .
408
409   getbitsandparityandsto | get bits and parity and stopbits :
410   | data bits := conf.bits par stop MOD 8 ;
411   | parity := (conf.bits par stop DIV 8) MOD 4 ;
412   | stop := (conf.bits par stop DIV 32) MOD 4 ;
413   | chose key (data bits, 7, bits per char, " Bits", PROC bits ok) ;
414   | IF data bits >= 0
415   | THEN chose key (parity, 2, parities, " parity", PROC parity ok)
416   |     chose key (stop, 2, stopbits, " Stopbits", PROC stopbits
417   |         ok);
418   |     conf.bits par stop := data bits + 8 * parity + 32 * stop
419   |     ELSE conf.bits par stop := data bits
420   |     FI .
421
422   getprotocol        | get protocol :
423   | chose key (conf.flow control, 10, flow modes,
424   |     "", PROC flow mode ok) .
425
426   getbuffersize     | get buffer size :
427   | IF dev type is transparent
428   | THEN chose buffer size
429   | ELSE conf.inbuffer size := std inbuffer size
430   | FI .

```

```

430 devtypeistransparent | dev type is transparent :
431 | conf.dev type = "transparent" .
432 |
433 chosebuffersize | chose buffer size :
434 | REP
435 | IF conf.inbuffer size = 16 CAND yes ("normaler Puffer")
436 | THEN LEAVE chose buffer size
437 | FI ;
438 | conf.inbuffer size := 512 ;
439 | IF yes ("grosser Puffer")
440 | THEN LEAVE chose buffer size
441 | FI ;
442 | conf.inbuffer size := 16
443 | PER .
444 |
445 | ENDPROC get configuration from user ;
446 |

```

```

447 execonfiguration ..... | PROC exec configuration :
448 |
449 | setup
450 |
451 | ENDPROC exec configuration ;
452 |

```

```

453 setup ..... | PROC setup :
454 |
455 | conf := old ("configuration") ;
456 | continue (configuration channel) ;
457 | FOR channel no FROM 1 UPTO max edit terminal REP
458 | set up channel (channel no, conf (channel no))
459 | PER ;
460 | continue (operators channel) .
461 |
462 | ENDPROC set up ;
463 |

```

```

464 setupchannel ..... | PROC set up channel (INT CONST channel no, CONF CONST conf) :
465 |
466 | link (channel no, conf.dev type) ;
467 | baudrate (channel no, conf.baud) ;
468 | bits (channel no, conf.bits par stop) ;
469 | flow (channel no, conf.flow control) ;
470 | input buffer size (channel no, conf.inbuffer size) .
471 |
472 | ENDPROC setup channel ;
473 |
474 | ENDPACKET configurator single ;
475 |

```