


```
SSSSSSSS HH HH 000000 WW WW LL 000000 GGGGGGGG
SSSSSSSS HH HH 000000 WW WW LL 000000 GGGGGGGG
SS SS HH HH 00 00 WW WW LL 00 00 GG
SS SS HH HH 00 00 WW WW LL 00 00 GG
SS SS HH HH 00 00 WW WW LL 00 00 GG
SSSSSS HH HH HH HH HH HH 00 00 WW WW LL 00 00 GG
SSSSSS HH HH HH HH HH HH 00 00 WW WW LL 00 00 GG
SS SS HH HH 00 00 WW WW LL 00 00 GG GGGGGG
SS SS HH HH 00 00 WW WW LL 00 00 GG GGGGGG
SS SS HH HH 00 00 WWW WWW LL 00 00 GG GG
SSSSSSSS HH HH 000000 WW WW LLLLLLLLLL 000000 GGGGGG
SSSSSSSS HH HH 000000 WW WW LLLLLLLLLL 000000 GGGGGG
.....
.....
.....
.....
```

```
LL I I I I I SSSSSSSS
LL I I I I I SSSSSSSS
LL I I SSS
LL I I SSS
LL I I SSS
LL I I SSSSSS
LL I I SSSSSS
LL I I SS
LL I I SS
LL I I SS
LL I I SS
LLLLLLLLLL I I I I I SSSSSSSS
LLLLLLLLLL I I I I I SSSSSSSS
```

```
1 0001 0 module showlog (ident = 'V04-000', ! File: SHOWLOG.B32 Edit: RAS0319
2 0002 0 addressing_mode (external = general))=
3 0003 1 begin
4 0004 1
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
10 0010 1 * ALL RIGHTS RESERVED. *
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
17 0017 1 * TRANSFERRED. *
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
21 0021 1 * CORPORATION. *
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
25 0025 1 *
26 0026 1 *
27 0027 1 *****
28 0028 1
29 0029 1
30 0030 1 ++
31 0031 1
32 0032 1 FACILITY: Show logical names
33 0033 1
34 0034 1 ABSTRACT:
35 0035 1 This module implements the "SHOW LOGICAL" DCL command.
36 0036 1
37 0037 1 ENVIRONMENT:
38 0038 1 VAX native, user and kernel mode
39 0039 1
40 0040 1 AUTHOR: Peter D Gilbert CREATION DATE: 20-Dec-1983
41 0041 1
42 0042 1 MODIFIED BY:
43 0043 1
44 0044 1 V03-004 RAS0328 Ron Schaefer 25-Jul-1984
45 0045 1 Make SHOW LOGICAL more friendly in terms of space allocated;
46 0046 1 instead of asking for PAGEDYN, ask for the minimum of
47 0047 1 PAGEDYN and process free-space as determined by $GETJPI.
48 0048 1 Fix possible kernel-mode accvio on error path.
49 0049 1
50 0050 1 V03-003 RAS0319 Ron Schaefer 2-Jul-1984
51 0051 1 Add additional data structure cell to the RT structure.
52 0052 1 Since RT and NT are actually used in 2 facilities, they
53 0053 1 should really be globally defined.
54 0054 1
55 0055 1 V03-002 PDG0002 Peter D Gilbert 08-Apr-1984
56 0056 1
57 0057 1 Add LOG_V_EQUAL, which is set in CMP_LOG, and indicates there
```

```
58 0058 1 are two or more identical logical names in the same table.
59 0059 1 This flag indicates that, even if no other attributes are
60 0060 1 displayed, the access mode should be displayed to distinguish
61 0061 1 between these logical names.
62 0062 1
63 0063 1 Change DUMP_LOG so a depth of 0 is shown as " ", instead of '0'.
64 0064 1
65 0065 1 For recursive translations, avoid writing a blank line until we
66 0066 1 know it's needed to separate logical names (when NO names were
67 0067 1 found, the extra blank line looked very tacky).
68 0068 1
69 0069 1 V03-001 TMK0001 Todd M. Katz 04-Apr-1984
70 0070 1 Make the following changes:
71 0071 1
72 0072 1 1. I have changed this module to use the symbol LNMSC_MAXDEPTH
73 0073 1 to define the maximum logical name recursion depth instead
74 0074 1 of the local symbol RT_C_MAXDEP.
75 0075 1
76 0076 1 2. The name of the offset in the table header that contains the
77 0077 1 address of the protection structure associated with shareable
78 0078 1 logical name tables has been changed from LNMTSL_CHIP to
79 0079 1 LNMTSL_ORB to reflect the replacement of shareable logical
80 0080 1 name tables' CHIP protection templates by Object Rights
81 0081 1 Blocks.
82 0082 1
83 0083 1 3. Change the routine recursively_translate, so that it
84 0084 1 immediately exits whenever it is called with a depth equal
85 0085 1 to LNMSC_MAXDEPTH. Previously, this routine was exiting only
86 0086 1 when it was called with a depth greater than LNMSC_MAXDEPTH.
87 0087 1 However, since recursion depths are numbered from 0 to 9 and
88 0088 1 not from 1 to 10, this was resulting in names being
89 0089 1 recursively translated to a depth of eleven instead of to a
90 0090 1 depth of ten.
91 0091 1
92 0092 1 --
93 0093 1
94 0094 1 library 'sys$library:lib'; ! VAX/VMS system definitions
95 0095 1 library 'sys$library:xport';
96 0096 1 %if %declare(%quote $descriptor) %then undeclare %quote $descriptor; %fi
97 0097 1 require 'src$:showdef'; ! SHOW common definitions
98 0196 1
99 0197 1
100 0198 1 B L I S S E X T E N S I O N S
101 0199 1
102 0200 1 macro
103 0201 1 $check_(x) = (local s: initial(x); if not .s then signal_stop(.s)) %,
104 0202 1
105 0203 1 Macros for offset, position, size, and extension.
106 0204 1
107 0205 1 base = 0.0,0.0 %,
108 0206 1 b0 = 0.8,0 %,
109 0207 1 b1 = 8.8,0 %,
110 0208 1 l = 0.32,0 %.
111 0209 1
112 0210 1 This macro is similar to the Bliss %fieldexpand lexical function.
113 0211 1 However, it works with either field names, or o,p,s,e macros.
114 0212 1
```

SHOWLOG
V04-000

L 15
16-Sep-1984 01:28:17
14-Sep-1984 12:09:37

VAX-11 Bliss-32 V4.0-742
[CLIUTL.SRC]SHOWLOG.B32;1

Page 3
(1)

```
: 115      M 0213 1  %fieldexpand(a,b,c,d,e) =
: 116      M 0214 1  %if %length eql 1 %then %fieldexpand(a) %else
: 117      M 0215 1  %if %length eql 2 %then %fieldexpand(a,b) %else
: 118      M 0216 1  %if %length eql 4 %then a,b,c,d %else
: 119      M 0217 1  %if %length eql 5 %then
: 120      M 0218 1  %if e eql 0 %then a %else
: 121      M 0219 1  %if e eql 1 %then b %else
: 122      M 0220 1  %if e eql 2 %then c %else
: 123      M 0221 1  %if e eql 3 %then d %else
: 124      M 0222 1  %error('Invalid parameter to %fieldexpand_')
: 125      M 0223 1  %fi %fi %fi %fi %else
: 126      M 0224 1  %error('Wrong number of arguments to %fieldexpand_')
: 127      M 0225 1  %fi %fi %fi %fi %;
```

```
129 0226 1 |
130 0227 1 | BLOCK TO CONTROL RECURSIVE TABLE NAME TRANSLATION
131 0228 1 |
132 0229 1 |      31      25 24      16      9 8 7      0
133 0230 1 | R5 -> +-----+-----+-----+-----+
134 0231 1 |      | TRIES      | DEPTH      |      | T C | ACCESS MODE |
135 0232 1 |      +-----+-----+-----+-----+
136 0233 1 |      |      |      |      |      |      |
137 0234 1 |      |      |      |      |      |      |
138 0235 1 |      |      |      |      |      |      |
139 0236 1 |      |      |      |      |      |      |
140 0237 1 |      |      |      |      |      |      |
141 0238 1 |      |      |      |      |      |      |
142 0239 1 |      |      |      |      |      |      |
143 0240 1 |      |      |      |      |      |      |
144 0241 1 |      |      |      |      |      |      |
145 0242 1 |      |      |      |      |      |      |
146 0243 1 |      |      |      |      |      |      |
147 0244 1 |      |      |      |      |      |      |
148 0245 1 |      |      |      |      |      |      |
149 0246 1 |      |      |      |      |      |      |
150 0247 1 |      |      |      |      |      |      |
151 0248 1 |      |      |      |      |      |      |
152 0249 1 |      |      |      |      |      |      |
153 0250 1 |      |      |      |      |      |      |
154 0251 1 |      |      |      |      |      |      |
155 0252 1 |      |      |      |      |      |      |
156 0253 1 |      |      |      |      |      |      |
157 0254 1 |      |      |      |      |      |      |
158 0255 1 |      |      |      |      |      |      |
159 0256 1 |      |      |      |      |      |      |
160 0257 1 |      |      |      |      |      |      |
161 0258 1 |      |      |      |      |      |      |
162 0259 1 |      |      |      |      |      |      |
163 0260 1 |      |      |      |      |      |      |
164 0261 1 |      |      |      |      |      |      |
165 0262 1 |      |      |      |      |      |      |
166 0263 1 |      |      |      |      |      |      |
167 0264 1 |      |      |      |      |      |      |
168 0265 1 |      |      |      |      |      |      |
169 0266 1 |      |      |      |      |      |      |
170 0267 1 |      |      |      |      |      |      |
171 0268 1 |      |      |      |      |      |      |
172 0269 1 |      |      |      |      |      |      |
173 0270 1 | R3 -> +-----+-----+-----+-----+
174 0271 1 |      | HASH FUNCTION VALUE OR 0 |      | C | ACCESS MODE |
175 0272 1 |      +-----+-----+-----+-----+
176 0273 1 |      |      |      |      |      |      |
177 0274 1 |      |      |      |      |      |      |
178 0275 1 |      |      |      |      |      |      |
179 0276 1 |      |      |      |      |      |      |
180 0277 1 |      |      |      |      |      |      |
181 0278 1 |      |      |      |      |      |      |
182 0279 1 |      |      |      |      |      |      |
183 0280 1 |      |      |      |      |      |      |
184 0281 1 |      |      |      |      |      |      |
185 0282 1 |      |      |      |      |      |      |
```

LNMSC_MAXDEPTH LONGWORDS TO SERVE AS STACK

T IS SET TO INHIBIT FURTHER RECURSION

Literal

rt_c_maxtries= 255; ! Max number of tries

\$unit_field rt_fields =

set

rt_w_r5= [\$bits(16)], ! Access mode and flags

\$overlay(rt_w_r5)

rt_b_acmode= [\$bits(8)], ! Access mode

rt_b_flags= [\$bits(8)], ! Flags

\$continue

\$overlay(rt_b_flags)

rt_v_case= [\$bit], ! Caseless flag

rt_v_term= [\$bit], ! Inhibit recursion flag

\$continue

rt_b_depth= [\$bits(8)], ! Recursion depth

rt_b_tries= [\$bits(8)], ! Recursion tries

rt_l_cachept= [\$address], ! Address of cache entry

rt_a_stack= [\$bits(0)] ! Start of recursion stack

tes;

Literal rt_k_size = \$field_set_units + lnmsc_maxdepth * %upval;

macro rt_block = \$unit_block(rt_k_size) field(rt_fields) %;

```
171 0268 1 |
172 0269 1 | BLOCK TO CONTROL NAME TRANSLATION
173 0270 1 |
174 0271 1 |      31      25 24      16 15      8 7      0
175 0272 1 | R3 -> +-----+-----+-----+-----+
176 0273 1 |      |      |      |      |      |      |
177 0274 1 |      |      |      |      |      |      |
178 0275 1 |      |      |      |      |      |      |
179 0276 1 |      |      |      |      |      |      |
180 0277 1 |      |      |      |      |      |      |
181 0278 1 |      |      |      |      |      |      |
182 0279 1 |      |      |      |      |      |      |
183 0280 1 |      |      |      |      |      |      |
184 0281 1 |      |      |      |      |      |      |
185 0282 1 |      |      |      |      |      |      |
```

F IS SET TO INHIBIT FURTHER RECURSION

```

: 186      0283 1 $unit_field nt_fields =
: 187      0284 1   set
: 188      0285 1   nt_w_r5=          [$bits(16)],    ! Access mode and flags
: 189      0286 1   $overlay(nt_w_r5)
: 190      0287 1   nt_b_acmode=      [$bits(8)],    ! Access mode
: 191      0288 1   nt_b_flags=       [$bits(8)],    ! Flags
: 192      0289 1   $continue
: 193      0290 1   $overlay(nt_b_flags)
: 194      0291 1   nt_v_case=        [$bit],        ! Caseless flag
: 195      0292 1   $continue
: 196      0293 1   nt_w_hash=        [$bits(16)],    ! Hash function or 0
: 197      0294 1   nt_l_namlen=      [$integer],     ! Length of name
: 198      0295 1   nt_l_namadr=      [$address],     ! Address of name
: 199      0296 1   nt_l_tabid=       [$address],     ! Table header address or 0
: 200      0297 1   nt_l_thread=      [$address]     ! Lnmb pointer
: 201      0298 1   tes;
: 202      0299 1 literal nt_k_size   = $field set units;
: 203      0300 1 macro   nt_block   = $unit_block(nt_k_size) field(nt_fields) %;

```

```
: 205 0301 1 ! Define linkages for the routines to lock and unlock the logical name
: 206 0302 1 ! table mutex, as well as the match routine and the protection checking
: 207 0303 1 ! routine.
: 208 0304 1
: 209 0305 1 linkage
: 210 0306 1     lnk_presearch = jsb (      ! RO = RETURN STATUS
: 211 0307 1         register=1,      ! R1 = HASH TABLE ADDRESS (IF TABLE HEADER ADDRESS IS MISSING)
: 212 0308 1         register=3,      ! R3 = ADDRESS OF NAME TRANSLATION (NT) BLOCK
: 213 0309 1         register=1,      ! R1 = ADDRESS OF LOGICAL NAME BLOCK THAT CONTAINS MATCH.
: 214 0310 1     ): nopreserve(0,1,2) preserve(3,4,5) notused(6,7,8,9,10,11);
: 215 0311 1 linkage
: 216 0312 1     lnk_contsearch = jsb (      ! RO = RETURN STATUS
: 217 0313 1         register=3,      ! R3 = ADDRESS OF NAME TRANSLATION (NT) BLOCK
: 218 0314 1         register=1,      ! R1 = ADDRESS OF LOGICAL NAME BLOCK THAT CONTAINS MATCH.
: 219 0315 1     ): nopreserve(0,1,2) preserve(3,4,5) notused(6,7,8,9,10,11);
: 220 0316 1 linkage
: 221 0317 1     lnk_check_prot = jsb (      ! RO = RETURN STATUS
: 222 0318 1         register=1,      ! R1 = ADDRESS OF LOGICAL NAME TABLE HEADER.
: 223 0319 1         register=2,      ! R2 = ACCESS MASK TO USE FOR THE CHECK.
: 224 0320 1         register=4,      ! R4 = CURRENT PCB ADDR
: 225 0321 1         register=5,      ! R5 = ACCESS MODE OF ACCESSOR
: 226 0322 1     ): nopreserve(2) preserve(1,3,4,11) notused(6,7,8,9,10);
: 227 0323 1 linkage
: 228 0324 1     lnk_hash = jsb (      ! RO = RETURNED HASH VALUE
: 229 0325 1         register=0,      ! RO = LENGTH OF LOGICAL NAME STRING
: 230 0326 1         register=1,      ! R1 = ADDRESS OF LOGICAL NAME STRING
: 231 0327 1     ): nopreserve(2,3) preserve(4,5) notused(6,7,8,9,10,11);
: 232 0328 1 linkage
: 233 0329 1     lnk_setup = jsb (
: 234 0330 1         register=2,      ! R2 = LENGTH OF TABLE NAME STRING
: 235 0331 1         register=3,      ! R3 = ADDRESS OF TABLE NAME STRING
: 236 0332 1         register=5,      ! R5 = ADDRESS OF RECURSION TABLE SEARCH CONTROL BLOCK
: 237 0333 1         register=1,      ! R1 = ADDRESS OF LOGICAL NAME TABLE HEADER
: 238 0334 1     ): nopreserve(2,3) preserve(4,5) notused(6,7,8,9,10,11);
: 239 0335 1 linkage
: 240 0336 1     lnk_table = jsb (
: 241 0337 1         register=5,      ! R5 = ADDRESS OF RECURSION TABLE SEARCH CONTROL BLOCK
: 242 0338 1         register=1,      ! R1 = ADDRESS OF LOGICAL NAME TABLE HEADER
: 243 0339 1     ): nopreserve(2,3) preserve(4,5) notused(6,7,8,9,10,11);
: 244 0340 1 linkage
: 245 0341 1     lnk_lock = jsb (
: 246 0342 1         register=4,      ! R4 = CURRENT PCB ADDR
: 247 0343 1     ): nopreserve(1,2,3) notused(5,6,7,8,9,10,11);
: 248 0344 1 linkage
: 249 0345 1     lnk_match_name = jsb (
: 250 0346 1         register=2,      ! R2 = LENGTH OF STRING TO TRY MATCHING
: 251 0347 1         register=3,      ! R3 = ADDRESS OF STRING TO TRY MATCHING
: 252 0348 1         register=4,      ! R4 = LENGTH OF WILDCARD STRING
: 253 0349 1         register=5,      ! R5 = ADDRESS OF WILDCARD STRING
: 254 0350 1     ): nopreserve(1,2,3,4,5) notused(6,7,8,9,10,11);
: 255 0351 1 linkage
: 256 0352 1     lnk_compare = jsb (
: 257 0353 1         register=0,
: 258 0354 1         register=1
: 259 0355 1     ): nopreserve(0,1,2,3) preserve(4,5) notused(6,7,8,9,10,11);
: 260 0356 1
: 261 0357 1 ! Some linkages that we don't use
```

```

: 262 0358 1 |
: 263 0359 1 | linkage
: 264 0360 1 |     lnk_searchlog = jsb (      ! R0 = RETURN STATUS
: 265 0361 1 |         register=0,          ! R0 = LENGTH OF LOGICAL NAME STRING
: 266 0362 1 |         register=1,          ! R1 = ADDRESS OF LOGICAL NAME STRING
: 267 0363 1 |         register=2,          ! R2 = LENGTH OF TABLE NAME STRING
: 268 0364 1 |         register=3,          ! R3 = ADDRESS OF TABLE NAME STRING
: 269 0365 1 |         register=5;          ! R5 = SEARCH ACCESS MODE IN LOW BYTE, CASELESS FLAG IN BIT 8.
: 270 0366 1 |         register=1           ! R1 = ADDRESS OF LOGICAL NAME BLOCK THAT CONTAINS MATCH.
: 271 0367 1 |     ): preserve(2,3,4,5) notused(6,7,8,9,10,11);
: 272 0368 1 | linkage
: 273 0369 1 |     lnk_firsttab = jsb (      ! R0 = RETURN STATUS
: 274 0370 1 |         register=1,          ! R1 = SEARCH ACCESS MODE IN LOW BYTE, CASELESS FLAG IN BIT 8.
: 275 0371 1 |         register=2,          ! R2 = LENGTH OF TABLE NAME STRING
: 276 0372 1 |         register=3;          ! R3 = ADDRESS OF TABLE NAME STRING
: 277 0373 1 |         register=1           ! R1 = ADDRESS OF LOGICAL NAME TABLE HEADER
: 278 0374 1 |     ): nopreserve(2,3) preserve(4,5) notused(6,7,8,9,10,11);
: 279 0375 1 | linkage
: 280 0376 1 |     lnk_lookup = jsb (
: 281 0377 1 |         register=0,          ! R0 = HASH CODE TABLE NAME STRING (OR 0 IF NOT KNOWN)
: 282 0378 1 |         register=2,          ! R2 = LENGTH OF TABLE NAME STRING
: 283 0379 1 |         register=3;          ! R3 = ADDRESS OF TABLE NAME STRING
: 284 0380 1 |         register=5;          ! R5 = ADDRESS OF RECURSION TABLE SEARCH CONTROL BLOCK
: 285 0381 1 |         register=1           ! R1 = ADDRESS OF LOGICAL NAME BLOCK
: 286 0382 1 |     ): nopreserve(2,3) preserve(5) notused(4,6,7,8,9,10,11);

```

```

: 288      0383 1  ! External declarations
: 289      0384 1  !
: 290      0385 1  external routine
: 291      0386 1      cli$present,
: 292      0387 1      cli$get_value,
: 293      0388 1      lib$get_vm,
: 294      0389 1      lib$free_vm,
: 295      0390 1      lib$scopy_dxdx,
: 296      0391 1      str$free_dx,
: 297      0392 1      lnm$check_prot:      lnk_check_prot,
: 298      0393 1      lnm$contsearch:      lnk_contsearch, ! ???
: 299      0394 1      lnm$presearch:      lnk_presearch, ! ???
: 300      0395 1      lnm$setup:      lnk_setup, ! ???
: 301      0396 1      lnm$table:      lnk_table, ! ???
: 302      0397 1      lnm$lockr:      lnk_lock,
: 303      0398 1      lnm$unlock:      lnk_lock,
: 304      0399 1      lnm$searchlog:      lnk_searchlog,
: 305      0400 1      lnm$firsttab:      lnk_firsttab,
: 306      0401 1      lnm$hash:      lnk_hash,
: 307      0402 1      lnm$lookup:      lnk_lookup, ! ???
: 308      0403 1      show$write_line,
: 309      0404 1      fmg$ma*ch_name:      lnk_match_name;
: 310      0405 1
: 311      0406 1  external literal
: 312      0407 1      show$_notran;
: 313      0408 1
: 314      0409 1  external
: 315      0410 1      sgn$gl_pagedyn,
: 316      0411 1      lnm$al_hashtbl:      vector[2], ! Addr of sys & proc hash tables
: 317      0412 1      lnm$al_dirtbl:      vector[2], ! Addr of sys & proc directory lnmbs
: 318      0413 1      ctl$gl_lnmhash,
: 319      0414 1      lnm$gl_htblsizp,
: 320      0415 1      lnm$gl_htblsizs,
: 321      0416 1      lnm$system_directory,
: 322      0417 1      ctl$gl_pcb:      ref $bblock;
```

```
0418 1 FLAG_BLOCK
0419 1
0420 1 This defines various flags that are used.
0421 1 Most of the flags are gotten directly from CLI calls.
0422 1
0423 1 $unit_field flag_fields =
0424 1   set
0425 1     flag_v_system      = [$bit],      ! True if /SYSTEM specified
0426 1     flag_v_group       = [$bit],      ! True if /GROUP specified
0427 1     flag_v_jc          = [$bit],      ! True if /JOB specified
0428 1     flag_v_proc        = [$bit],      ! True if /PROCESS specified
0429 1     flag_v_normal      = [$bit],      ! This means not /STRUCTURE
0430 1     flag_v_descend     = [$bit],      ! True if descendants needed
0431 1     flag_v_full        = [$bit],      ! True if /FULL specified
0432 1     flag_v_recurse     = [$bit],      ! True to show recursive translations
0433 1     flag_v_found       = [$bit],      ! True if any translations found
0434 1   tes;
0435 1 literal flag_k_size   = $field_set units;
0436 1 macro   flag_block    = $unit_block(flag_k_size) field(flag_fields) %;
0437 1
0438 1
0439 1
0440 1 CMTBL_BLOCK
0441 1
0442 1 This is used to store a table or a logical name from the command line
0443 1 (the roots of these linked lists are cmdtbl_list and cmdnam_list).
0444 1 For table names, the cmdtbl_l_table field points to a list of
0445 1 table blocks that are relevant/correspond to the particular table name.
0446 1 For logical names, the cmdtbl_l_table field is unused.
0447 1
0448 1 $unit_field cmdtbl_fields =
0449 1   set
0450 1     cmdtbl_l_next      = [$address],
0451 1     cmdtbl_l_table     = [$address],
0452 1     cmdtbl_q_name      = [$bytes(dsc$%k_d_bln)]
0453 1   tes;
0454 1 literal cmdtbl_k_size = $field_set units;
0455 1 macro   cmdtbl_block  = $unit_block(cmdtbl_k_size) field(cmdtbl_fields) %;
0456 1
0457 1
0458 1 TABLE_BLOCK
0459 1
0460 1 This stores the information we will display about a table.
0461 1 It also provides a tree structure for the tables (via the table_l_next
0462 1 and table_l_child fields).
0463 1 The table_l_log field points to a linked list of log_blocks.
0464 1
0465 1 $unit_field table_fields =
0466 1   set
0467 1     table_l_next       = [$address],
0468 1     table_l_child      = [$address],
0469 1     table_l_byteslm    = [$integer],  ! Same as lnmt$%l_byteslm
0470 1     table_l_bytes      = [$integer],  ! Same as lnmt$%l_bytes
0471 1     table_l_log        = [$address],  ! Pointer to log_block
0472 1     table_b_acmode     = [$byte],     ! Same as lnmb$%b_acmode
0473 1     table_b_flags      = [$byte],     ! Same as lnmt$%b_flags
0474 1     table_t_name       = [$byte]      ! Ascic name of the table
```

```
381 0475 1 tes;
382 0476 1 literal table_k_size = $field set units;
383 0477 1 macro table_block = $unit_block(table_k_size) field(table_fields) %;
384 0478 1
385 0479 1
386 0480 1 LOG_BLOCK
387 0481 1
388 0482 1 This holds the information for one logical name.
389 0483 1 These blocks are actually variable length --
390 0484 1 from log_t_name through the end of the log_block, the contents are the
391 0485 1 same as lnmb$t_name through the end of a lnmb block.
392 0486 1
393 0487 1 $unit_field log_fields =
394 0488 1 set
395 0489 1 log_l_next = [$address],
396 0490 1 log_l_table = [$address], ! Pointer to table_block
397 0491 1 log_b_flags = [$byte], ! Same as lnmb$b_flags
398 0492 1 $overlay(log_b_flags)
399 0493 1 log_v_dummy = [$bits(7)],
400 0494 1 log_v_equal = [$bit], ! An identical logical is in this table
401 0495 1 $continue
402 0496 1 log_b_acmode = [$byte], ! Same as lnmb$b_acmode
403 0497 1 log_t_name = [$byte] ! Same as lnmb$t_name
404 0498 1 tes;
405 0499 1 literal log_k_size = $field set units;
406 0500 1 macro log_block = $unit_block(log_k_size) field(log_fields) %;
407 0501 1
408 0502 1
409 0503 1 KNOWN_BLOCK
410 0504 1
411 0505 1 This block is used to store which logical name tables are currently
412 0506 1 known (to avoid displaying the table and contained names more than
413 0507 1 once -- consider /TABLE=(LNMB$DCL_LOGICAL,LNMB$PROCESS) ).
414 0508 1
415 0509 1 A lookup on this structure uses known_l_lnmth as a key, and gives
416 0510 1 known_l_table as the result.
417 0511 1
418 0512 1 $unit_field known_fields =
419 0513 1 set
420 0514 1 known_l_next = [$address],
421 0515 1 known_l_lnmth = [$address], ! Address of the lnmth block
422 0516 1 known_l_table = [$address] ! Pointer to the table_block
423 0517 1 tes;
424 0518 1 literal known_k_size = $field set units;
425 0519 1 macro known_block = $unit_block(known_k_size) field(known_fields) %;
426 0520 1
427 0521 1
428 0522 1 OWN STORAGE
429 0523 1
430 0524 1 own
431 0525 1 vm_size, ! Remaining size of VM
432 0526 1 vm_addr, ! Starting address of VM
433 0527 1 saved_fp, ! Saved FP from kernel mode
434 0528 1 known_list: ref known_block, ! List of "known" tables
435 0529 1 cmdtbl_list: ref cmdtbl_block, ! Tables from CLI
436 0530 1 cmdnam_list: ref cmdtbl_block, ! Logical names from CLI
437 0531 1 flags: ! Various flags
```

```
438      0532 1      acmode:          byte:          ! Integrity mode
439      0533 1
440      0534 1
441      0535 1      ! Macro for a routine to re-call itself in kernel mode
442      0536 1      ! (if it's not already running in kernel mode).
443      0537 1
444      0538 1      ! The sole rationale for this is to make the code more readable.
445      0539 1
446      0540 1      ! This macro also saves the FP of the kernel mode code.
447      0541 1      ! The idea is to provide an easy way to return out of kernel mode.
448      0542 1      ! And while we're at it, lock the logical name tables for read access.
449      0543 1
450      0544 1      macro
451      M 0545 1      runs_in_kernel_(rtn) =
452      M 0546 1      begin
453      M 0547 1      builtin ap, movpsl, fp;
454      M 0548 1      local psl: $bblock[%upval];
455      M 0549 1      movpsl(PSL);
456      M 0550 1      if .psl[psl$v_curmod] eql psl$c_kernel
457      M 0551 1      then
458      M 0552 1      saved_fp = .fp
459      M 0553 1      else
460      M 0554 1      begin
461      M 0555 1      local status;
462      M 0556 1      status = $cmkrnl(routin = rtn, arglst = .ap);
463      M 0557 1      saved_fp = 0;
464      M 0558 1      return .status;
465      M 0559 1      end;
466      M 0560 1      lnm$lockr(.ctl$gl_pcb;
467      0561 1      end %;
468      0562 1
469      0563 1      ! Macro to write a blank line
470      0564 1
471      0565 1      bind
472      0566 1      blank_line = %ascid'';
473      0567 1      macro
474      M 0568 1      write_blank_line(z) =
475      M 0569 1      util_output(blank_line)
476      0570 1      %;
477      0571 1
478      0572 1      routine util_output(ctrstr, p1) =
479      0573 2      begin
480      0574 2      return show$write_line(.ctrstr, p1);
481      0575 1      end;
```

```
.TITLE SHOWLOG
.IDENT \V04-000\
.PSECT $PLITS,NOWRT,NOEXE,2
.BKLB 0
.LONG 17694720
.ADDRESS P.AAB
.PSECT $OWNS,NOEXE,2
```

```
010E0000 00000 P.AAB:
00000000 00000 P.AAA:
00000000 00004
```

```
...
...
```

H 16
16-Sep-1984 01:28:17
14-Sep-1984 12:09:37

VAX-11 Bliss-32 V4.0-742
[CLIUTL.SRC]SHOWLOG.B32;1

00000 VM_SIZE:.BLKB 4
00004 VM_ADDR:.BLKB 4
00008 SAVED_FP:
.BLKB 4
0000C KNOWN_LIST:
.BLKB 4
00010 CMDTBL_LIST:
.BLKB 4
00014 CMDNAM_LIST:
.BLKB 4
00018 FLAGS: .BLKB 2
0001A ACMODE: .BLKB 1

BLANK_LINE= P.AAA
.EXTRN CLISPRESENT, CLISGET_VALUE
.EXTRN LIB\$GET_VM, LIB\$FREE_VM
.EXTRN LIB\$COPY_DXDX, STR\$FREE1_DX
.EXTRN LNMSCHECK_PROT, LNMSCONTSEARCH
.EXTRN LNMSPRESEARCH, LNMSSETUP
.EXTRN LNMSTABLE, LNMSLOCKR
.EXTRN LNMSUNLOCK, SHOW\$WRITE_LINE
.EXTRN FMG\$MATCH_NAME, SHOW\$NOTRAN
.EXTRN SGN\$GL_PAGEDYN, LNMSAL_HASHTBL
.EXTRN LNMSAL_DIRTBL, CTL\$GL_PCB
.PSECT \$CODE\$,NOWRT,2

0000 00000 UTIL_OUTPUT:

	08	AC	9F	00002	.WORD	Save nothing	
	04	AC	DD	00005	PUSHAB	P1	
00000000G 00		02	FB	00008	PUSHL	CTRSTR	
		04	0000F		CALLS	#2, SHOW\$WRITE_LINE	
					RET		

: 0572
: 0574
:
:
: 0575

; Routine Size: 16 bytes, Routine Base: \$CODE\$ + 0000

```

: 483 0576 1 : Routine to allocate some memory.
: 484 0577 1 : A large chunk of memory has already been allocated (by lib$get_vm).
: 485 0578 1 : This routine simply grabs a little piece of that large chunk.
: 486 0579 1 :
: 487 0580 1 : Because the amount of memory we allocated is based on sgn$gl_pagedyn, and we
: 488 0581 1 : are pretty good about not storing information twice, we should never run out
: 489 0582 1 : of this memory. However, if we do, we're ready for it (see below).
: 490 0583 1 :
: 491 0584 1 routine util_allocate(size) =
: 492 0585 2 begin
: 493 0586 2
: 494 0587 2 if (vm_size = .vm_size - .size) geq 0
: 495 0588 2 then
: 496 0589 2 return .vm_addr + .vm_size
: 497 0590 2 else if
: 498 0591 2 .saved_fp eql 0
: 499 0592 2 then
: 500 0593 2 begin
: 501 0594 2 : We aren't in kernel mode. Signal insufficient memory.
: 502 0595 2 :
: 503 0596 2 signal(show$_insvirmem);
: 504 0597 2 return 0;
: 505 0598 2 end
: 506 0599 2
: 507 0600 2 else
: 508 0601 2 begin
: 509 0602 2 : We are in kernel mode. Return a status to the caller, by
: 510 0603 2 : setting FP to that of our first kernel mode frame and returning.
: 511 0604 2 :
: 512 0605 2 builtin
: 513 0606 2 fp;
: 514 0607 2 lnm$unlock(.ctl$gl_pcb);
: 515 0608 2 set_ipl(0);
: 516 0609 2 fp = .saved_fp;
: 517 0610 2 return show$_insvirmem;
: 518 0611 2
: 519 0612 2 end;
: 520 0613 1 end;
```

```

                                003C 00000 UTIL_ALLOCATE:
                                .WORD Save R2,R3,R4,R5
                                MOVAB VM_SIZE, R5
                                SUBL2 SIZE, VM_SIZE
                                BLSS 1$
                                ADDL3 VM_SIZE, VM_ADDR, R0
                                RET
                                08 A5 D5 00013 1$: TSTL SAVED_FP
                                10 12 00016 BNEQ 2$
                                00000000G 00 007812F2 8F DD C0018 PUSHL #7869170
                                01 FB 0001E CALLS #1, LIB$SIGNAL
                                50 D4 00025 CLRL R0
                                04 00027 RET
                                54 00000000G 00 D0 00028 2$: MOVL CTL$GL_PCB, R4
```

```

: 0584
: 0587
: 0589
: 0590
: 0591
: 0597
: 0598
: 0608
```

```
00000000G 00 16 0002F JSB LNMSUNLOCK
12 00 00 DA 00035 MTPR #0, #18
5D 08 A5 D0 00038 MOVL SAVED FP, FP
50 007812F2 8F D0 0003C MOVL #7869T70, R0
04 00043 RET
```

```
: 0609
: 0610
: 0611
: 0613
```

; Routine Size: 68 bytes, Routine Base: \$CODE\$ + 0010

```
: 521 0614 1
: 522 0615 1
: 523 0616 1 ! Routine to re-allocate some memory.
: 524 0617 1 ! The reallocation is like removing stuff from a stack.
: 525 0618 1 ! The parameter is the address of where we want to back up to.
: 526 0619 1
: 527 0620 1 routine util_reallocate(addr): novalue =
: 528 0621 2 begin
: 529 0622 2 vm_size = .addr - .vm_addr;
: 530 0623 1 end;
```

0000 00000 UTIL_REALLOCATE:

```
0000' CF 04 AC 0000' CF C3 00002 .WORD Save nothing
04 0000B SUBL3 VM_ADDR, ADDR, VM_SIZE
RET
```

```
: 0620
: 0622
: 0623
```

; Routine Size: 12 bytes, Routine Base: \$CODE\$ + 0054

```
: 531 0624 1
: 532 0625 1
: 533 0626 1 ! Routine to manipulate pointers.
: 534 0627 1 ! All the parameters are by reference, except the last which is by value.
: 535 0628 1 ! Slide the rightmost parameter to the left, and so on. Return the value
: 536 0629 1 ! of the leftmost parameter.
: 537 0630 1
: 538 0631 1 routine util_slide =
: 539 0632 2 begin
: 540 0633 2 builtin
: 541 0634 2 actualcount,
: 542 0635 2 actualparameter;
: 543 0636 2 macro
: 544 0637 2 swap_(x,y) = (local t; t = .x; x = .y; y = .t) %:
: 545 0638 2 local
: 546 0639 2 x;
: 547 0640 2 x = actualparameter(actualcount());
: 548 0641 2 decr i from actualcount()-1 to 1 do
: 549 0642 2 swap_(actualparameter(.i), x);
: 550 0643 2 return .x;
: 551 0644 1 end;
```

```
000C 00000 UTIL_SLIDE:
50          6C 9A 00002      .WORD      Save R2,R3
53          6C40 D0 00005      MOVZBL    (AP), R0
50          6C 9A 00009      MOVZBL    (AP)[R0], X
11 11 0000C      BRB          2$
51          6C40 D0 0000E 1$:  MOVL      (AP)[I], R1
52          61 D0 00012      MOVL      (R1), T
51          6C40 D0 00015      MOVL      (AP)[I], R1
61          53 D0 00019      MOVL      X, (R1)
53          52 D0 0001C      MOVL      T, X
EC          50 F5 0001F 2$:  SOBGTR    I, 1$
50          53 D0 00022      MOVL      X, R0
04 00025      RET
```

; Routine Size: 38 bytes, Routine Base: \$CODE\$ + 0060

```
: 552      0645 1
: 553      0646 1
: 554      0647 1 ! Routine to add another cmdtbl_block to a linked list.
: 555      0648 1 !
: 556      0649 1 routine add_to_cmd_list(
: 557      0650 1     p_root,           ! Address of the root of the linked list
: 558      0651 1     name: ref $block  ! String to be added to the list
: 559      0652 1     ): novalue =
: 560      0653 2     begin
: 561      0654 2     bind
: 562      0655 2     root = .p_root: ref cmdtbl_block;
: 563      0656 2     local
: 564      0657 2     ptr: ref cmdtbl_block;
: 565      0658 2     ptr = util_allocate(cmdtbl_k_size);
: 566      0659 2     ch$move(dsc$kd_bln, name[base_], ptr[cmdtbl_q_name]);
: 567      0660 2     ptr = util_slide(ptr[cmdtbl_l_next], root, ptr[base_]);
: 568      0661 1     end;
```

```
007C 00000 ADD_TO_CMD_LIST:
82 AF          10 DD 00002      .WORD      Save R2,R3,R4,R5,R6
56            01 FB 00004      PUSHL     #16
08 A6          50 D0 00008      CALLS    #1, UTIL_ALLOCATE
08 BC          08 28 0000B      MOVZBL    R0, PTR
04            56 DD 00011      MOVZBL    #8, @NAME, 8(PTR)
            AC DD 00013      PUSHL     PTR
            56 DD 00016      PUSHL     P_ROOT
            03 FB 00018      PUSHL     PTR
            BE AF          50 D0 0001C      CALLS    #3, UTIL_SLIDE
            56            04 0001F      MOVL      R0, PTR
            RET
```

; Routine Size: 32 bytes, Routine Base: \$CODE\$ + 0086

```

: 569      0662 1
: 570      0663 1
: 571      0664 1 ! This clever little routine reverses a linked list.
: 572      0665 1 ! This simplifies building the list, and traversing the list.
: 573      0666 1
: 574      0667 1 routine reverse_list(
: 575      0668 1     p_root,      ! Address of the root of the linked list
: 576      0669 1     next       ! Byte offset of the 'next' field in a node
: 577      0670 1 ): novalue =
: 578      0671 2     begin
: 579      0672 2     bind
: 580      0673 2         root = .p_root: ref $bblock;
: 581      0674 2     local
: 582      0675 2         ptr: ref $bblock initial(0);
: 583      0676 2     while .root neq 0 do
: 584      0677 2         ptr = util_slide(root, root[.next,l_], ptr[base_]);
: 585      0678 2         root = ptr[base_];
: 586      0679 1     end;
```

```

                                0000 00000 REVERSE_LIST:
                                .WORD      Save nothing
                                CLRL      PTR
                                04 BC D5 00004 1$: TSTL @P_ROOT
                                11 13 00007 BEQL 2$
                                50 DD 00009 PUSHL PTR
                                7E 04 BC 08 AC C1 0000B ADDL3 NEXT, @P_ROOT, -(SP)
                                04 AC DD 00011 PUSHL P_ROOT
                                A2 AF 03 FB 00014 CALLS #3, UTIL_SLIDE
                                EA 11 00018 BRB 1$
                                04 BC 50 D0 0001A 2$: MOVL PTR, @P_ROOT
                                04 0001E RET
```

```

: 0667
: 0673
: 0676
: 0677
:
:
:
: 0678
: 0679
```

; Routine Size: 31 bytes, Routine Base: \$CODE\$ + 00A6

```

: 587      0680 1
: 588      0681 1 routine sort_list(      ! list[cmdtbl_l_table], %fieldexpand(table_l_next,0));
: 589      0682 1     p_root,      ! Address of the root of the linked list
: 590      0683 1     next,      ! Byte offset of the 'next' field in a node
: 591      0684 1     compare     ! Address of a routine to compare nodes
: 592      0685 1 ): novalue =
: 593      0686 2     begin
: 594      0687 2     !
: 595      0688 2     ! Alphabetize the list pointed to by p_root
: 596      0689 2     !
: 597      0690 2     bind
: 598      0691 2         root = .p_root: ref $bblock;
: 599      0692 2     bind routine
: 600      0693 2         compare_rtn = .compare: lnk_compare;
: 601      0694 2     local
: 602      0695 2         sort: ref $bblock initial(0);
: 603      0696 2     while root[base_] neq 0 do
: 604      0697 3         begin
```

```

: 605      0698      3
: 606      0699      3
: 607      0700      3
: 608      0701      3
: 609      0702      3
: 610      0703      3
: 611      0704      3
: 612      0705      3
: 613      0706      4
: 614      0707      4
: 615      0708      4
: 616      0709      4
: 617      0710      4
: 618      0711      3
: 619      0712      3
: 620      0713      2
: 621      0714      2
: 622      0715      1

Grab one item from root list, and insert in it sort list

local
  r: ref $bblock
  p: ref vector[1];
r = sort = .next;
while (p = r[.next,l_]; r = .p[0]) neq 0 do
  begin
    if compare_rtn( r[base_], root[base_] ) geq 0
    then
      exitloop;
    end;
  r = util_slide(root, root[.next,l_], 0);
  r = util_slide(r[.next,l_], p[0], -r[base_]);
end;
root = sort[base_];
end;
```

```

                                00FC 0000 SORT_LIST:
                                .WORD
                                Save R2,R3,R4,R5,R6,R7
                                57      7E D4 00002 CLRL SORT
                                57      6E 9E 00004 MOVAB SORT, R7
                                55      08 AC C2 00007 SUBL2 NEXT, R7
                                55      04 BC D0 0000B 1$: MOVL @P_ROOT, R5
                                54      3A 13 0000F BEQL 4$
                                54      57 D0 00011 MOVL R7, R
                                54      08 AC C1 00014 2$: ADDL3 NEXT, R, P
                                54      66 D0 00019 MOVL (P), R
                                50      0A 13 0001C BEQL 3$
                                50      54 7D 0001E MOVQ R, R0
                                0C      BC 16 00021 JSB @COMPARE
                                50      D5 00024 TSTL R0
                                50      EC 19 00026 BLSS 2$
                                50      7E D4 00028 3$: CLRL -(SP)
                                08 BC45 9F 0002A PUSHAB @NEXT[R5]
                                04 AC DD 0002E PUSHL P_ROOT
                                FF65 CF 03 FB 00031 CALLS #3, UTIL_SLIDE
                                54      50 D0 00036 MOVL R0, R
                                54      DD 00039 PUSHL R
                                56      DD 0003B PUSHL P
                                08 BC44 9F 0003D PUSHAB @NEXT[R]
                                FF55 CF 03 FB 00041 CALLS #3, UTIL_SLIDE
                                54      50 D0 00046 MOVL R0, R
                                04      C0 11 00049 BRB 1$
                                04      BC 6E DC 0004B 4$: MOVL SORT, @P_ROOT
                                04      04 0004F RET
                                0681
                                0693
                                0704
                                0696
                                0704
                                0705
                                0707
                                0711
                                0712
                                0696
                                0714
                                0715
```

; Routine Size: 80 bytes, Routine Base: \$CODE\$ + 00C5

; 623 0716 1
; 624 0717 1

```

: 625      0718 1  This recursive routine does a preorder tree traversal
: 626      0719 1  (visit node, visit left subtree, visit right subtree).
: 627      0720 1
: 628      0721 1  To reduce the amount of stack space required, the traversal
: 629      0722 1  is done partly by recursion, partly by iteration.
: 630      0723 1
: 631      0724 1 routine visit_tree(
: 632      0725 1     root: ref $bblock,      ! Root of the tree
: 633      0726 1     loffset,              ! Byte offset of the 'left' field in a node
: 634      0727 1     roffset,              ! Byte offset of the 'right' field in a node
: 635      0728 1     action,                ! Address of an action routine
: 636      0729 1     depth                  ! Current depth of the traversal
: 637      0730 1 ): novalue =
: 638      0731 2 begin
: 639      0732 2 local
: 640      0733 2     ptr: ref $bblock;
: 641      0734 2     ptr = root = .roffset;
: 642      0735 2 while (ptr = .ptr[.roffset, l_]) neq 0 do
: 643      0736 2     begin
: 644      0737 2         (.action)(ptr[base_], .depth);
: 645      0738 2         visit_tree(.ptr[.loffset, l_], .loffset, .roffset, .action, .depth+1);
: 646      0739 2     end;
: 647      0740 1 end;
```

```

                                000C 00000 VISIT_TREE:
                                .WORD      Save R2,R3
                                MOVAB      ROOT, R0
                                SUBL3      ROFFSET, R0, PTR
                                ADDL3      #1, DEPTH, R3
                                ADDL3      ROFFSET, PTR, R0
                                MOVL      (R0), PTR
                                BEQL      2$
                                PUSHL      DEPTH
                                PUSHL      PTR
                                CALLS      #2, @ACTION
                                PUSHL      R3
                                MOVQ      ROFFSET, -(SP)
                                PUSHL      LOFFSET
                                ADDL3      LOFFSET, PTR, R0
                                PUSHL      (R0)
                                CALLS      #5, VISIT_TREE
                                BRB        1$
                                04 00039 2$: RET
                                : 0724
                                : 0734
                                : 0738
                                : 0735
                                : 0737
                                : 0738
                                : 0735
                                : 0740
```

; Routine Size: 58 bytes, Routine Base: \$CODE\$ + 0115

```

649 0741 1 : This routine calls the CLI and builds a list of cmdtbl_blocks to hold
650 0742 1 : the table names specified by the user.
651 0743 1
652 0744 1 routine get_tables =
653 0745 2 begin
654 0746 2 local
655 0747 2 desc: $block[dsc$d_bln],
656 0748 2 root: ref cmdtbl_block initial(0);
657 0749 2 macro
658 0750 2 add_(x) = add_to_cmd_list( root, x ) %;
659 0751 2
660 0752 2 if not .flags[flag_v_normal]
661 0753 2 then
662 0754 2 begin
663 0755 2
664 0756 2 : for /STRUCTURE, just specify the directories,
665 0757 2 : and ask for descendants.
666 0758 2
667 0759 2 : Note that the /STRUCTURE qualifier should not be used with
668 0760 2 : any of the /TABLE, /PROCESS, /JOB, /GROUP or /SYSTEM qualifiers.
669 0761 2
670 0762 2 add_( %ascid'LNMSPROCESS DIRECTORY' );
671 0763 2 add_( %ascid'LNMS$SYSTEM DIRECTORY' );
672 0764 2 flags[flag_v_descend] = true;
673 0765 2 end
674 0766 2 else
675 0767 2 begin
676 0768 2
677 0769 2 : Get all the relevant table names, in the following order:
678 0770 2 : names from the /TABLE qualifier
679 0771 2 : names from the /PROCESS, /JOB, /GROUP and /SYSTEM qualifiers
680 0772 2 : if still no names, use LNMSDCL_LOGICAL
681 0773 2
682 0774 2 while
683 0775 2 begin
684 0776 2 $init_dyndesc(desc);
685 0777 2 cli$get_value(%ascid'table', desc[base_])
686 0778 2 end
687 0779 2 do
688 0780 2 add_(desc[base_]);
689 0781 2 if .flags[flag_v_proc] then add_( %ascid 'LNMSPROCESS' );
690 0782 2 if .flags[flag_v_job] then add_( %ascid 'LNMS$JOB' );
691 0783 2 if .flags[flag_v_group] then add_( %ascid 'LNMS$GROUP' );
692 0784 2 if .flags[flag_v_system] then add_( %ascid 'LNMS$SYSTEM' );
693 0785 2 if .root eql 0 then add_( %ascid 'LNMSDCL_LOGICAL' );
694 0786 2 end;
695 0787 2
696 0788 2 reverse_list( root, %fieldexpand_(cmdtbl_l_next,0) );
697 0789 2
698 0790 2 return .root;
699 0791 2 end;
```

.PSECT SPLITS,NOWRT,NOEXE,2

52 49 44 5f 53 53 45 43 4f 52 50 24 4d 4e 4c 00008 P.AAD: .ASCII \LNMSPROCESS_DIRECTORY\<0><0><0>

```
00 00 00 59 52 4F 54 43 45 00017
010E0015 00020 P.AAC: .LONG 17694741
00000000' 00024 .ADDRESS P.AAD
45 52 49 44 5F 4D 45 54 53 59 53 24 4D 4E 4C 00028 P.AAF: .ASCII \LNMS\SYSTEM_DIRECTORY\
59 52 4F 54 43 00037
010E0014 0003C P.AAE: .LONG 17694740
00000000' 00040 .ADDRESS P.AA-
00 00 00 65 6C 62 61 74 00044 P.AAH: .ASCII \table\ 0><0><0>
010E0005 0004C P.AAG: .LONG 17694725
00000000' 00050 .ADDRESS P.AAH
00 53 53 45 43 4F 52 50 24 4D 4E 4C 00054 P.AAJ: .ASCII \LNMSPROCESS\<0>
010E000B 00060 P.AAI: .LONG 17694731
00000000' 00064 .ADDRESS P.AAJ
00 42 4F 4A 24 4D 4E 4C 00068 P.AAL: .ASCII \LNMSJOB\<0>
010E0007 00070 P.AAK: .LONG 17694727
00000000' 00074 .ADDRESS P.AAL
00 00 00 50 55 4F 52 47 24 4D 4E 4C 00078 P.AAN: .ASCII \LNMSGROUP\<0><0><0>
010E0009 00084 P.AAM: .LONG 17694729
00000000' 00088 .ADDRESS P.AAN
00 00 4D 45 54 53 59 53 24 4D 4E 4C 0008C P.AAP: .ASCII \LNMSYSTEM\<0><0>
010E000A 00098 P.AAO: .LONG 17694730
00000000' 0009C .ADDRESS P.AAP
4C 41 43 49 47 4F 4C 5F 4C 43 44 24 4D 4E 4C 000A0 P.AAR: .ASCII \LNMSDCL_LOGICAL\<0>
00 000AF
010E000F 000B0 P.AAQ: .LONG 17694735
00000000' 000B4 .ADDRESS P.AAR
```

.PSECT \$CODE\$,NOWRT,2

001C 00000 GET_TABLES:

```
54 0000' CF 9E 00002 .WORD Save R2,R3,R4
53 0000' CF 9E 00007 MOVAB FLAGS, R4
52 FF27 CF 9E 0000C MOVAB P.AAC, R3
5E 08 C2 00011 MOVAB ADD_TO_CMD_LIST, R2
7E D4 00014 SUBL2 #8, -SP
16 64 04 E0 00016 CLRL ROOT
53 DD 0001A BBS #4, FLAGS, 1$
04 AE 9F 0001C PUSHL R3
62 02 FB 0001F PUSHAB ROOT
1C A3 9F 00022 CALLS #2, ADD_TO_CMD_LIST
04 AE 9F 00025 PUSHAB P.AAE
62 02 FB 00028 PUSHAB ROOT
64 20 88 0002B CALLS #2, ADD_TO_CMD_LIST
67 11 0002E B!SB2 #32, FLAGS
04 AE 020E0000 8F D0 00030 1$: MOVL #34471936, DESC
08 AE D4 00038 CLRL DESC+4
04 AE 9F 0003B PUSHAB DESC
2C A3 9F 0003E PUSHAB P.AAG
00000000G 00 02 FB 00041 CALLS #2, CLISGET_VALUE
08 50 E9 00048 BLBC R0, 2$
04 AE 9F 0004B PUSHAB DESC
04 AE 9F 0004E PUSHAB ROOT
62 02 FB 00051 CALLS #2, ADD_TO_CMD_LIST
DA 11 00054 BRB 1$
```

0744

0745
0752
0762

0763

0764
0752
0776

0777

0780

09	64	40	03	E1	00056	2\$:	BBC	#3, FLAGS, 3\$	: 0781
		04	A3	9F	0005A		PUSHAB	P.AAI	
	62		02	FB	00060		PUSHAB	ROOT	
09	64	50	02	E1	00063	3\$:	CALLS	#2, ADD_TO_CMD_LIST	: 0782
		04	A3	9F	00067		BBC	#2, FLAGS, -4\$	
	62		02	FB	0006D		PUSHAB	P.AAK	
09	64	64	AE	9F	0006A		PUSHAB	ROOT	: 0783
		04	02	FB	0006D		CALLS	#2, ADD_TO_CMD_LIST	
	62		01	E1	00070	4\$:	BBC	#1, FLAGS, -5\$	: 0783
		04	A3	9F	00074		PUSHAB	P.AAM	
	62		AE	9F	00077		PUSHAB	ROOT	
	09	78	02	FB	0007A		CALLS	#2, ADD_TO_CMD_LIST	: 0784
		04	64	E9	0007D	5\$:	BLBC	FLAGS, 8\$	
	62		A3	9F	00080		PUSHAB	P.AAO	
		04	AE	9F	00083		PUSHAB	ROOT	
			02	FB	00086		CALLS	#2, ADD_TO_CMD_LIST	: 0785
			6E	D5	00089	6\$:	TSTL	ROOT	
			0A	12	0008B		BNEQ	7\$	
		0090	C3	9F	0008D		PUSHAB	P.AAQ	
		04	AE	9F	00091		PUSHAB	ROOT	
	62		02	FB	00094		CALLS	#2, ADD_TO_CMD_LIST	: 0788
		04	7E	D4	00097	7\$:	CLRL	-(SP)	
			AE	9F	00099		PUSHAB	ROOT	
20	A2		02	FB	0009C		CALLS	#2, REVERSE_LIST	: 0790
	50		6E	D0	000A0		MOVL	ROOT, R0	: 0791
			04	000A3			RET		

; Routine Size: 164 bytes, Routine Base: \$CODE\$ + 014F

```

700 0792 1
701 0793 1
702 0794 1 ! This routine calls the CLI and builds a list o cmdtbl_blocks to hold
703 0795 1 ! the logical names specified by the user.
704 0796 1
705 0797 1 routine get_lognam =
706 0798 2   begin
707 0799 2     local
708 0800 2       desc: $block[dsc$d_bln],
709 0801 2       root: ref cmdtbl_block initial(0);
710 0802 2
711 0803 2   macro
712 0804 2     add_(x) = add_to_cmd_list( root, x ) %;
713 0805 2   while
714 0806 2     begin
715 0807 2       $init_dyndesc(desc);
716 0808 2       cli$get_value(%ascid'logname', desc[base_])
717 0809 2     end
718 0810 2   do
719 0811 2     begin
720 0812 2       ! If the last character is a colon, remove it.
721 0813 2       !
722 0814 2       ! if ch$rchar(.desc[dsc$w_length]+.desc[dsc$a_pointer]) eql ':'
723 0815 2       then
724 0816 2         begin
725 0817 2           local
726 0818 2

```

```

: 727      0819  4      tmp:      vector[2];
: 728      0820  4      tmp[0] = .desc[dsc$w_length] - 1;
: 729      0821  4      tmp[1] = .desc[dsc$a_pointer];
: 730      0822  4      lib$scopy_dxdx(tmp[0], desc[base_]);
: 731      0823  3      end;
: 732      0824  3      add_(desc[base_]);
: 733      0825  2      end;
: 734      0826  2      if .root eql 0      then add ( %ascid '*' );
: 735      0827  2      reverse_list( root, %fieldexpand_(cmdtbl_l_next,0) );
: 736      0828  2
: 737      0829  2      return .root;
: 738      0830  1      end;
```

.PSECT \$PLITS\$,NOWRT,NOEXE,2

```

00 65 6D 61 6E 67 6F 6C 000B8 P.AAT: .ASCII \logname\<0>
                                010E0007 000C0 P.AAS: .LONG 17694727
                                00000000' 000C4 .ADDRESS P.AAT
                                00 00 00 2A 000C8 P.AAV: .ASCII \*\<0><0><0>
                                010E0001 000CC P.AAU: .LONG 17694721
                                00000000' 000D0 .ADDRESS P.AAV
```

.PSECT \$CODE\$,NOWRT,2

```

                                0000 00000 GET_LOGNAM:
                                .WORD      Save nothing
                                5E          10 C2 00002  SUBL2      #16, SP
                                7E          D4 00005  CLRL       ROOT
                                OC AE 020E0000 8F D0 00007 1$:  MOVL      #34471936, DESC
                                10          AE D4 0000F  CLRL      DESC+4
                                OC          AE 9F 00012  PUSHAB    DESC
                                0000'      CF 9F 00015  PUSHAB    P.AAS
00000000G 00          02 FB 00019  CALLS     #2, CLISGET_VALUE
                                34          50 E9 00020  BLBC      R0, 3$
                                50          OC AE 3C 00023  MOVZWL    DESC, R0
                                50          10 AE C0 00027  ADDL2     DESC+4, R0
                                3A          60 91 0002B  CMPB      (R0), #58
                                04 AE          1A 12 0002E  BNEQ      2$
                                04 AE          3C 0003C  MOVZWL    DESC, TMP
                                08 AE          D7 00035  DECL      TMP
                                08 AE          D0 00038  MOVL      DESC+4, TMP+4
                                OC          AE 9F 0003D  PUSHAB    DESC
                                08          AE 9F 00040  PUSHAB    TMP
00000000G 00          02 FB 00043  CALLS     #2, LIB$SCOPY_DXD
                                OC          AE 9F 0004A 2$:  PUSHAB    DESC
                                04          AE 9F 0004D  PUSHAB    ROOT
                                FE3E CF          02 FB 00050  CALLS     #2, ADD_TO_CMD_LIST
                                BO          11 00055  BRB       1$
                                6E          D5 00057 3$:  TSTL      ROOT
                                OC          12 00059  BNEQ      4$
                                0000'      CF 9F 0005B  PUSHAB    P.AAU
                                04          AE 9F 0005F  PUSHAB    ROOT
                                FE2C CF          02 FB 00062  CALLS     #2, ADD_TO_CMD_LIST
```

```
G 1
16-Sep-1984 01:28:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:09:37 [CL]UTL.SRC]SHOWLOG.B32;1
```

: 0827

FE42	CF	04	7E	D4	00067	48:	CLRL	-(SP)
	50		AE	9F	00069		PUSHAB	ROOT
			02	FB	0006C		CALLS	#2, REVERSE_LIST
			6E	D0	00071		MOVL	ROOT, R0
				04	00074		RET	

0829
0830

; Routine Size: 117 bytes, Routine Base: \$CODES + 01F3

```

: 740      0831 1 ! Little routine to check the protection on a lnmth block
: 741      0832 1
: 742      0833 1 routine check_prot(x: ref $bblock) =
: 743      0834 2   begin
: 744      0835 2     literal
: 745      0836 2     pro_mask = arm$m_read;           ! Protection mask
: 746      0837 2     if not .x[lnmth$y_shareable] then return true
: 747      0838 2     else if .x[lnmth$y_directory] then return true      ! ???
: 748      0839 2     else if .x[lnmth$l_orb] eql 0 then return true
: 749      0840 2     else return lnm$check_prot(x[base_], pro_mask, .ctl$gl_pcb, 0);
: 750      0841 1   end;
```

```

                                083C 00000 CHECK_PROT:
                                .WORD      Save R2,R3,R4,R5,R11
                                51          04 AC D0 00002      MOVL      X, R1
                                09          61 E9 00006      BLBC      (R1), 1$
05                                61          01 E0 00009      BBS       #1, (R1), 1$
                                05          A1 D5 0000D      TSTL      5(R1)
                                04          12 00010      BNEQ      2$
                                50          01 D0 00012 1$:    MOVL      #1, R0
                                04          04 00015      RET
                                55          D4 00016 2$:    CLRL      R5
                                54 00000000G 00 D0 00018      MOVL      CTL$GL_PCB, R4
                                52          01 D0 0001F      MOVL      #1, R2
                                00          16 00022      JSB       LNM$CHECK_PROT
                                04          04 00028      RET
```

; Routine Size: 41 bytes. Routine Base: \$CODE\$ + 0268

```

: 751      0842 1
: 752      0843 1
: 753      0844 1 routine recheck_tables: novalue =
: 754      0845 2   begin
: 755      0846 2
: 756      0847 2     When the code reenters kernel mode, there is always the possibility
: 757      0848 2     that some logical name table found by get_relevant_tables got deleted,
: 758      0849 2     or is no longer accessible (which is okay per se, since we really don't
: 759      0850 2     need to find the lnmth block again).
: 760      0851 2
: 761      0852 2     However, it is possible that one of the lnmth blocks got deleted and
: 762      0853 2     reused as another lnmth block. This means that we may be displaying
: 763      0854 2     logical names that the user has no business seeing!
: 764      0855 2
: 765      0856 2     This routine prevents the above security hole, by the following method:
: 766      0857 2     For each table in known_list, we find the table again (by name or by
: 767      0858 2     calling for each lnmth and checking the known_l_lnmth value), and verify
: 768      0859 2     that the user has read access to the table (this prevents the security
: 769      0860 2     hole; we should also check that the tables have the same name and
: 770      0861 2     attributes). If it doesn't check out, the known_l_lnmth value is zeroed;
: 771      0862 2     since lnmbl table is never zero, we'll not find any more logical names
: 772      0863 2     for this table.
: 773      0864 2
```

```

: 0833
: 0837
: 0838
: 0839
:
: 0840
:
: 0841
```



```

: 831      0922 3
: 832      0923 2      end;
: 833      0924 1      end;

```

```

                                00FC 00000 RECHECK_TABLES:
                                .WORD Save R2,R3,R4,R5,R6,R7
                                SUBL2 #48, SP
                                MOVAB AN_RT_BLOCK, RT
                                MOVW #3, (RT)
                                BISB2 #1, 1(RT)
                                MOVAB KNOWN_LIST, KNOWN
                                MOVL (KNOWN), KNOWN
                                BEQL 6$
                                TSTL 4(KNOWN)
                                BEQL 1$
                                MOVL 8(KNOWN), TABLE
                                MOVAB 23(TABLE), R3
                                MOVZBL 22(TABLE), R2
                                JSB LNM$SETUP
                                MOVL R0, STATUS
                                MOVL R1, R6
                                BLBS STATUS, 3$
                                CMPL STATUS, #884
                                BNEQ 4$
                                DECB 2(RT)
                                BRB 5$
                                CMPL LNMTH, 4(KNOWN)
                                BNEQ 5$
                                PUSHL LNMTH
                                CALLS #1, CHECK_PROT
                                BLBS R0, 1$
                                CLRL 4(KNOWN)
                                BRB 1$
                                MNEGB #1, 3(RT)
                                JSB LNM$TABLE
                                BRB 2$
                                RET
                                01 5E      30 C2 00002
                                55      6E 9E 00005
                                65      03 B0 00008
                                A5      01 88 0000B
                                54      CF 9E 0000F
                                54      64 D0 00014 1$:
                                04      4E 13 00017
                                50      A4 D5 00019
                                53      F6 13 0001C
                                52      A4 D0 0001E
                                00000000G 50 0A 00026
                                57      00 16 0002A
                                56      50 D0 00030 2$:
                                0E      51 D0 00033
                                00000374 0E 57 E8 00036
                                8F      57 D1 00039
                                02      14 12 00040
                                A5      97 00042
                                14 11 00045
                                04 A4      56 D1 00047 3$:
                                0E 12 00048
                                56 DD 0004D
                                84 AF      01 FB 0004F
                                BE      50 E8 00053
                                04 A4 D4 00056 4$:
                                B9 11 00059
                                03 A5      01 8E 0005B 5$:
                                00000000G 00 16 0005F
                                C9      11 00065
                                04 00067 6$:

```

; Routine Size: 104 bytes, Routine Base: \$CODE\$ + 0291

```
: 835      0925 1 : This routine does a lookup on the known list.
: 836      0926 1 : It looks up the address of a lnmth block:
: 837      0927 1 :     if found, it returns the address of the table_block,
: 838      0928 1 :     if not found, it returns zero.
: 839      0929 1 :
: 840      0930 1 routine lookup_table(lnmth_address) =
: 841      0931 2     begin
: 842      0932 2     local
: 843      0933 2     known: ref known_block;
: 844      0934 2     known = known_list - %fieldexpand (known_l_next,0);
: 845      0935 2     while (known = .known[known_l_next]) neq 0 do
: 846      0936 2     if .known[known_l_lnmth] eql .lnmth_address
: 847      0937 2     then
: 848      0938 2         return .known[known_l_table];
: 849      0939 2     return 0;
: 850      0940 1     end;
```

				0000 00000 LOOKUP_TABLE:		
					.WORD	Save nothing
	50	0000'	CF 9E 00002		MOVAB	KNOWN_LIST, KNOWN
	50		60 D0 00007 1\$:		MOVL	(KNOWN), KNOWN
			0C 13 0000A		BEQL	2\$
04	AC	04	A0 D1 0000C		CML	4(KNOWN), LNMTH_ADDRESS
			F4 12 00011		BNEQ	1\$
	50	08	A0 D0 00013		MOVL	8(KNOWN), R0
			04 00017		RET	
			50 D4 00018 2\$:		CLRI	R0
			04 0001A		RET	

```
: 0930
: 0934
: 0935
:
: 0936
:
: 0938
:
: 0939
: 0940
```

: Routine Size: 27 bytes, Routine Base: \$CODE\$ + 02F9

```
: 851      0941 1
: 852      0942 1
: 853      0943 1 : This routine adds a lnmth block to the list of known tables.
: 854      0944 1 :
: 855      0945 1 routine add_to_known(
: 856      0946 1     lnmth: ref $bblock
: 857      0947 1     ) =
: 858      0948 2     begin
: 859      0949 2
: 860      0950 2     Find whether we already know about this lnmth entry
: 861      0951 2     If we already know about it (or don't want the user to know about it,
: 862      0952 2     because of protection) then return 0,
: 863      0953 2     Otherwise, return a pointer to the table_block
: 864      0954 2
: 865      0955 2     local
: 866      0956 2     known: ref known_block;
: 867      0957 2
: 868      0958 2
: 869      0959 2     Check protection on the table
: 870      0960 2
: 871      0961 2     if not check_prot(lnmth[base_]) then return 0;
```

```

872 0962
873 0963
874 0964
875 0965
876 0966
877 0967
878 0968
879 0969
880 0970
881 0971
882 0972
883 0973
884 0974
885 0975
886 0976
887 0977
888 0978
889 0979
890 0980
891 0981
892 0982
893 0983
894 0984
895 0985
896 0986
897 0987
898 0988
899 0989
900 0990
901 0991
902 0992
903 0993
904 0994
905 0995
906 0996
907 0997

: See if it's already on the known list
: If so, return a zero.
if lookup_table(lnmth[base_]) neq 0 then return 0;

begin
bind lnmb = .lnmth[lnmth$l_name]: $bblock;
local table: ref table_block;

: Check the access mode on the table
if .lnmb[lnmb$b_acmode] gtru .acmode then return 0;

: Create and initialize a table_block
table = util_allocate(table k_size + .lnmb[lnmb$t_name]);
ch$move(1+.lnmb[lnmb$t_name], lnmb[lnmb$t_name], table[table_t_name]);
table[table_b_acmode] = .lnmb[lnmb$b_acmode];
table[table_b_flags] = .lnmth[lnmth$b_flags];
table[table_l_bytes] = .lnmth[lnmth$l_bytes];
table[table_l_byteslm] = .lnmth[lnmth$l_byteslm];

: Create and initialize a known_block
: Add the known_block to the list of known tables
known = util_allocate(known k_size);
known[known_l_lnmth] = lnmth[base_];
known[known_l_table] = table[base_];
util_slide(known[known_l_next], known_list, known[base_]);
return table[base_];
end;

end;
```

```

                                01FC 00000 ADD_TO_KNOWN:
                                .WORD      Save R2,R3,R4,R5,R6,R7,R8
                                58          04      AC      D0 00002      MOVL      LNMTH, R8
                                58          DD 00006      PUSHL     R8
FF47 CF          01      FB 00008      CALLS    #1, CHECK_PROT
                                50          E9 0000D      BLBC      R0, $
                                58          DD 00010      PUSHL     R8
CF AF          01      FB 00012      CALLS    #1, LOOKUP_TABLE
                                50          D5 00016      TSTL      R0
                                55          12 00018      BNEQ      1$
                                0000* 56          09      A8 D0 0001A      MOVL      9(R8), R6
                                CF          0B      A6 91 0001E      CMPB     11(R6), ACMODE
                                49          1A 00024      BGTRU     1$
                                7E          11      A6 9A 00026      MOVZBL   17(R6), -(SP)
                                6E          17      C0 0002A      ADDL2    #23, (SP)
FCCA CF          01      FB 0002D      CALLS    #1, UTIL_ALLOCATE
                                :
```

```

: 0945
: 0961
:
: 0967
:
: 0971
: 0976
: 0980
:
```

		57		50	D0	00032	MOVL	R0, TABLE			
		50	11	A6	9A	00035	MOVZBL	17(R6), R0	:	0981	
				50	D6	00039	INCL	R0	:		
16	A7	11	A6	50	28	0003B	MOVCL	R0, 17(R6), 22(TABLE)	:		
		14	A7	0B	A6	90	00041	MOVB	11(R6), 20(TABLE)	:	0982
		15	A7		68	90	00046	MOVB	(R8), 21(TABLE)	:	0983
		08	A7	1D	A8	7D	0004A	MOVQ	29(R8), 8(TABLE)	:	0985
					0C	DD	0004F	PUSHL	#12	:	0990
	FCA6	CF			01	FB	00051	CALLS	#1, UTIL_ALLOCATE	:	
	04	A0			58	D0	00056	MOVL	R8, 4(KNOWN)	:	0991
	08	A0			57	D0	0005A	MOVL	TABLE, 8(KNOWN)	:	0992
					50	DD	0005E	PUSHL	KNOWN	:	0993
				0000'	CF	9F	00060	PUSHAB	KNOWN_LIST	:	
					50	DD	00064	PUSHL	KNOWN	:	
	FCE1	CF			03	FB	00066	CALLS	#3, UTIL_SLIDE	:	
		50			57	D0	0006B	MOVL	TABLE, R0	:	0994
						04	0006E	RET		:	
					50	D4	0006F	1\$: C_LRL	R0	:	0997
					04	00071	RET			:	

; Routine Size: 114 bytes, Routine Base: \$CODE\$ + 0314

908	0998	1	
909	0999	1	
910	1000	1	! This little routine determines whether a lnmth table name
911	1001	1	matches any of the table names specified by the user.
912	1002	1	! It is called via the routine for_each_lnmth.
913	1003	1	!
914	1004	1	routine match_lnmth(lnmth: ref \$bblock, list: ref cmdtbl_block): novalue =
915	1005	2	begin
916	1006	2	bind name = list[cmdtbl_g_name]: \$bblock[dsc\$d_bln];
917	1007	2	bind lnmth = .lnmth[lnmth\$1_name]: \$bblock;
918	1008	2	local table: ref table_block;
919	1009	2	if fmg\$match_name(.lnmth[lnmth\$1_name], 1+lnmth[lnmth\$1_name],
920	1010	2	.name[dsc\$w_length], .name[dsc\$a_pointer]) then
921	1011	2	if (table = add_to_known(lnmth[base_])) neq 0
922	1012	2	then
923	1013	2	util_slide(table[table_l_next], list[cmdtbl_l_table], table[base_]);
924	1014	1	end;

				007C 00000 MATCH_LNMTH:				
				.WORD	Save R2,R3,R4,R5,R6	:	1004	
51	08	AC	08	C1 00002	ADDL3	#8, LIST, R1	:	1006
		56	04	AC D0 00007	MOVL	LNMT, R6	:	1007
		50	09	A6 D0 0000B	MOVL	9(R6), R0	:	
		53	12	A0 9E 0000F	MOVAB	18(R0), R3	:	1009
		55	04	A1 D0 00013	MOVL	4(R1), R5	:	
		54		61 3C 00017	MOVZWL	(R1), R4	:	
		52	11	A0 9A 0001A	MOVZBL	17(R0), R2	:	
			00000000G	00 16 0001E	JSB	FMG\$MATCH_NAME	:	
		19		50 E9 00024	BLBC	R0, 1\$	:	
				56 DD 00027	PUSHL	R6	:	1011

FF60	CF	01	FB	00029	CALLS	#1, ADD_TO_KNOWN
		50	D5	0002E	TSTL	TABLE
		0E	13	00030	BEQL	1\$
7E	08	50	DD	00032	PJSHL	TABLE
		04	C1	00034	ADDL3	#4, LIST, -(SP)
		50	DD	00039	PUSHL	TABLE
FC9A	CF	03	FB	0003B	CALLS	#3, UTIL_SLIDE
		04	00040	1\$:	RET	

1013

1014

; Routine Size: 65 bytes, Routine Base: \$CODE\$ + 0386

```
: 925      1015  1
: 926      1016  1
: 927      1017  1 ! This comparison routine is used for sorting table_blocks.
: 928      1018  1 ! It is called via the routine sort_list.
: 929      1019  1 !
: 930      1020  1 routine cmp_table( t1: ref table_block, t2: ref table_block ): lnk_compare =
: 931      1021  2   begin
: 932      1022  2   return ch$compare(
: 933      1023  2       .t1[table_t_name], 1+t1[table_t_name],
: 934      1024  2       .t2[table_t_name], 1+t2[table_t_name]);
: 935      1025  1   end;
```

				54	DD	00000	CMP_TABLE:		
							PUSHL	R4	1020
		53	16	A0	9A	00002	MOVZBL	22(T1), R3	1023
		52	16	A1	9A	00006	MOVZBL	22(T2), R2	1024
		54		01	D0	0000A	MOVL	#1, R4	1022
52	00	17	A0	53	2D	0000D	CMPC5	R3, 23(T1), #0, R2, 23(T2)	
				17	A1	00013			
				03	1A	00015	BGTRU	1\$	
		54		01	D9	00017	SBWC	#1, R4	
		50		54	D0	0001A	MOVL	R4, R0	
				10	BA	0001D	POPR	#^M<R4>	1025
				05	0001F		RSB		

; Routine Size: 32 bytes, Routine Base: \$CODE\$ + 03C7

```
: 936      1026  1
: 937      1027  1 ! This comparison routine is used for sorting log_blocks.
: 938      1028  1 ! It is called via the routine sort_list.
: 939      1029  1 !
: 940      1030  1 routine cmp_log( t1: ref log_block, t2: ref log_block ): lnk_compare =
: 941      1031  2   begin
: 942      1032  2   local
: 943      1033  2   s;
: 944      1034  2   s = ch$compare(
: 945      1035  2       .t1[log_t_name], 1+t1[log_t_name],
: 946      1036  2       .t2[log_t_name], 1+t2[log_t_name]);
: 947      1037  2   if .s eql 0
: 948      1038  2   then
```

```

: 949      1039 3      begin
: 950      1040 3      t1[log_v_equal] = true;
: 951      1041 3      t2[log_v_equal] = true;
: 952      1042 3      end;
: 953      1043 3      return .s;
: 954      1044 1      end;
```

```

                                30 BB 00000 CMP_LOG: PUSHR    #^M<R4,R5>
                                51 D0 00002      MOVL      R1, R5
                                50 D0 00005      PUSHL     R0
                                50 6E 00007      ADDL3     #10, T1, R0
                                51 51 00008      MOVZBL     (R0), R1
                                50 0A 9A 0000B      MOVZBL     10(T2), R0
                                54 01 D0 00012      MOVL      #1, R4
                                6E 0B C1 00015      ADDL3     #11, T1, -(SP)
50 00 9E 51 2D 00019      CMPC5     R1, a(SP)+, #0, R0, 11(T2)
                                0B A5 0001E
                                03 1A 00020      BGTRU      1$
                                54 01 D9 00022      SBWC       #1, R4
                                50 54 D0 00025 1$:      MOVL      R4, S
                                0D 12 00028      BNEQ       2$
                                51 6E 08 C1 0002A      ADDL3     #8, T1, R1
                                0B 61 8F 88 0002E      BISB2     #128, (R1)
                                08 A5 80 8F 88 00032      BISB2     #128, 8(T2)
                                5E 04 C0 00037 2$:      ADDL2     #4, SP
                                30 BA 0003A      POPR       #^M<R4,R5>
                                05 0003C      RSB
```

: Routine Size: 61 bytes, Routine Base: \$CODE\$ + 03E7

```

: 955      1045 1
: 956      1046 1
: 957      1047 1 ! This routine creates a log_block to hold the information about a logical name
: 958      1048 1 ! and links it to a table_block.
: 959      1049 1
: 960      1050 1 routine new_lnmb(lnmb: ref $bblock, table: ref table_block) =
: 961      1051 2   begin
: 962      1052 2     local
: 963      1053 2       size,
: 964      1054 2       log: ref log_block,
: 965      1055 2       lnmx: ref $bblock,
: 966      1056 2       funny: initial(true);
: 967      1057 2
: 968      1058 2     ! Do we really want this logical name?
: 969      1059 2
: 970      1060 2     if .lnmb[lnmb$b_acmode] gtru .acmode then return 0;
: 971      1061 2
: 972      1062 2     ! Find the end of the relevant information
: 973      1063 2     ! Also, make sure we find at least one logical name with a positive index.
: 974      1064 2
: 975      1065 2     lnmx = .lnmb[lnmb$t_name] + 1 + lnmb[lnmb$t_name];
: 976      1066 2     while not .lnmx[lnmx$u_xend] do
```

```

: 977      1067      3      begin
: 978      1068      3      if .lnmx[lnmx$b_index] geq 0 then funny = false;
: 979      1069      3      lnm = .lnmx[lnmx$st_xlation] + 1+lnmx[lnmx$st_xlation];
: 980      1070      3      end;
: 981      1071      2      if .funny and not .lnmb[lnmb$st_table] then return 0;
: 982      1072      2
: 983      1073      2
: 984      1074      2      ! Now that we know the beginning and end of the relevant information,
: 985      1075      2      ! allocate a log_block to hold it.
: 986      1076      2
: 987      1077      2      size = lnm[base_] - lnmb[lnmb$st_name] + 1+%fieldexpand_(lnmx$st_xend,0);
: 988      1078      2      log = util_allocate(log_k_size + .size);
: 989      1079      2      ch$move(.size, lnmb[lnmb$st_name], log[log_t_name]);
: 990      1080      2      util_slide(log[log_l_next], table[table_l_log], log[base_]);
: 991      1081      2      log[log_l_table] = table[base_];
: 992      1082      2      log[log_b_flags] = .lnmb[lnmb$b_flags];
: 993      1083      2      log[log_b_acmode] = .lnmb[lnmb$b_acmode];
: 994      1084      2      log[log_v_equal] = false;
: 995      1085      2      return log[base_];
: 996      1086      1      end;
```

```

                                00FC 00000 NEW_LNMB:
                                .WORD      Save R2,R3,R4,R5,R6,P7
                                52          01 D0 00002      MOVL      #1, FUNNY
                                57          04 AC D0 00005      MOVL      LNMB, R7
                                0000' CF      0B A7 91 00009      CMPB      11(R7), ACMODE
                                50          11 A7 9A 00011      BGTRU      5$
                                51          12 A740 9E 00015      MOVZBL     17(R7), R0
                                61          02 E0 0001A 1$:      MOVAB      18(R7)(R0), LNMX
                                01          A1 95 0001E      BBS       #2, (LNMX), 3$
                                02          19 00021      TSTB      1(LNMX)
                                52          D4 00023      BLSS      2$
                                50          04 A1 9A 00025 2$:      CLRL      FUNNY
                                51          05 A140 9E 00029      MOVZBL     4(LNMX), R0
                                EA          11 0002E      MOVAB      5(LNMX)(R0), LNMX
                                05          52 E9 00030 3$:      BRB       1$
                                42          10 A7 03 E1 00033      BLBC      FUNNY, 4$
                                52          11 A7 9E 00038 4$:      BBC       #3, 16(R7), 5$
                                52          51 C2 0003C      MOVAB      17(R7), R2
                                01          52 C3 0003F      SUBL2     LNMX, R2
                                FBA1 CF      0B A2 9F 00043      SUBL3     R2, #1, SIZE
                                56          01 FB 00046      PUSHAB    11(SIZE)
                                0A          A6 11 A7 52 28 0004B      CALLS     #1, UTIL_ALLOCATE
                                7E          08 AC 56 DD 00054      MOVL      R0, LOG
                                FBDA CF      03 FB 0005D      MOVCS     SIZE, 17(R7), 10(LOG)
                                04          A6 08 AC D0 00062      PUSHL     LOG
                                08          A6 10 A7 90 00067      ADDL3     #16, TABLE, -(SP)
                                09          A6 08 A7 90 0006C      PUSHL     LOG
                                08          A6 80 8F 8A 00071      CALLS     #3, UTIL_SLIDE
                                MOVL      TABLE, 4(LOG)
                                MOV      16(R7), 8(LOG)
                                MOV      11(R7), 9(LOG)
                                BICE2     #128, 8(LOG)
```

```
50          56 D0 00076      MOVL    LOG, R0
              04 00079      RET
50          D4 0007A 5$:    CLRL    R0
              04 0007C      RET
```

```
: 1085
: 1086
:
```

: Routine Size: 125 bytes, Routine Base: \$CODE\$ + 0424

```
997 1087 1
998 1088 1
999 1089 1 : This routine determines whether a lnmb block is relevant, and (if so) creates
1000 1090 1 : a log_block to hold the information, and links it to a table_block.
1001 1091 1 : It is called via the routine for_each_lnmb.
1002 1092 1
1003 1093 1 routine match_lnmb( lnmb: ref $bblock, dummy ): novalue =
1004 1094 2 begin
1005 1095 2 local
1006 1096 2 table: ref table_block;
1007 1097 2
1008 1098 2 : Verify that this is one of the desired logical names
1009 1099 2 : It must be in an acceptable integrity mode
1010 1100 2 : It must be in one of the specified tables
1011 1101 2 : it must match one of the logical names from the command line
1012 1102 2
1013 1103 2 begin
1014 1104 2 local
1015 1105 2 list: ref cmdtbl_block;
1016 1106 2
1017 1107 3 if .lnmb[lnmb$b_acmode] gtru .acmode then return;
1018 1108 3 if (table = lookup_table(.lnmb[lnmb$l_table])) eq 0 then return;
1019 1109 3
1020 1110 3 list = cmdnam_list - %fieldexpand(cmdtbl_l_next,0);
1021 1111 4 if ( while (list = .list[cmdtbl_l_next]) neq 0 do
1022 1112 5 begin
1023 1113 5 bind name = list[cmdtbl_q_name]: $bblock[dsc$d_bln];
1024 1114 5 if fmg$match_name(.lnmb[lnmb$t_name], 1+lnmb[lnmb$t_name],
1025 1115 5 .name[dsc$w_length], .name[dsc$a_pointer])
1026 1116 5 then
1027 1117 5 exitloop false;
1028 1118 4 end )
1029 1119 3 then
1030 1120 3 return;
1031 1121 2 end;
1032 1122 2
1033 1123 2 : Link the new lnmb block to the table
1034 1124 2 :
1035 1125 2 new_lnmb( lnmb[base_], table[base_] );
1036 1126 2
1037 1127 1 end;
```

```
03FC 00000 MATCH_LNMB:
57          04 AC D0 00002      .WORD    Save R2,R3,R4,R5,R6,R7,R8,R9
              MOVL    LNMB, R7
```

```
: 1093
: 1107
```

0000'	CF	0B	A7	91	00006	CMPB	11(R7), ACMODE	:	
			3F	1A	0000C	BGTRU	2\$	:	
		0C	A7	DD	0000E	PUSHL	12(R7)	:	1108
FE42	CF		01	FB	00011	CALLS	#1, LOOKUP_TABLE	:	
	59		50	D0	00016	MOVL	R0, TABLE	:	
			32	13	00019	BEQL	2\$	:	
	56	0000'	CF	9E	0001B	MOVAB	CMDNAM_LIST, LIST	:	1110
	58	12	A7	9E	00020	MOVAB	18(R7), R8	:	1114
	56		66	D0	00024	MOVL	(LIST), LIST	:	1111
			24	13	00027	BEQL	2\$	:	
	50	08	A6	9E	00029	MOVAB	8(LIST), R0	:	1113
	55	04	A0	D0	0002D	MOVL	4(R0), R5	:	1114
	54		60	3C	00031	MOVZWL	(R0), R4	:	
	53		58	D0	00034	MOVL	R8, R3	:	
	52	11	A7	9A	00037	MOVZBL	17(R7), R2	:	
		00000000G	00	16	0003B	JSB	FMG\$MATCH_NAME	:	
	E0		50	E9	00041	BLBC	R0, 1\$	:	
		0280	8F	BB	00044	PUSHR	#^M<R7,R9>	:	1125
FF36	CF		02	FB	00048	CALLS	#2, NEW_LNMB	:	
			04	0004D	2\$:	RET		:	1127

; Routine Size: 78 bytes. Routine Base: \$CODE\$ + 04A1

```
1038 1128 1
1039 1129 1
1040 1130 1
1041 1131 1 ! Get the descendants of the lnmtb block, and add them to the table block
1042 1132 1 !
1043 1133 1 routine get_descend(
1044 1134 1     table: ref table_block,
1045 1135 1     lnmtb: ref $bblock
1046 1136 1 ): novalue =
1047 1137 2     begin
1048 1138 2     local
1049 1139 2         sublnmtb: ref $bblock;
1050 1140 2         sublnmtb = lnmtb[lnmtb$l_child] - %fieldexpand(lnmtb$l_sibling,0);
1051 1141 2         while (sublnmtb = .sublnmtb[lnmtb$l_sibling]) neq 0 do
1052 1142 3             begin
1053 1143 3             local
1054 1144 3                 subtable: ref table_block;
1055 1145 3                 if (subtable = add_to_known(sublnmtb[base_])) neq 0
1056 1146 3                 then
1057 1147 4                     begin
1058 1148 4                         util_slide(subtable[table_l_next],
1059 1149 4                             table[table_l_child], subtable[base_]);
1060 1150 4                         get_descend(subtable[base_], sublnmtb[base_]);
1061 1151 4                     end
1062 1152 3                 else
1063 1153 4                     begin
1064 1154 4                         Note that we keep looking for descendant tables,
1065 1155 4                         even if we've run across one that we've either seen before,
1066 1156 4                         or have no privilege to read.
1067 1157 4                         Note: these get linked to the table, rather than the subtable.
1068 1158 4                     end
1069 1159 4                     get_descend(table[base_], sublnmtb[base_]);
1070 1160 4
```

```
: 1071      1161  3      end;
: 1072      1162  2      end;
: 1073      1163  2      sort_list( table[table_l_child], %fieldexpand_(table_l_next,0),
: 1074      1164  2      cmp_table );
: 1075      1165  1      end;
```

```
000C 00000 GET_DESCEND:
52      08      AC      15      04      C3      00002      .WORD      Save R2,R3
      52      15      A2      D0      00007      1$:      SUBL3      #4, LNMTH, SUBLNMTH
      2B      13      0000B      MOVL      21(SUBLNMTH), SUBLNMTH
      52      DD      0000D      BEQL      4$
      FE11      CF      01      FB      0000F      PUSHL      SUBLNMTH
      53      50      D0      00014      CALLS      #1, ADD TO KNOWN
      14      13      00017      MOVL      R0, SUBTABLE
      53      DD      00019      BEQL      2$
      7E      04      AC      04      C1      0001B      PUSHL      SUBTABLE
      FB4A      CF      53      DD      00020      ADDL3      #4, TABLE, -(SP)
      03      FB      00022      PUSHL      SUBTABLE
      52      DD      00027      CALLS      #3, UTIL_SLIDE
      53      DD      00029      PUSHL      SUBLNMTH
      05      11      0002B      BRB      3$
      52      DD      0002D      2$:      PUSHL      SUBLNMTN
      CA      AF      04      AC      DD      0002F      PUSHL      TABLE
      FE9C      CF      02      FB      00032      3$:      CALLS      #2, GET_DESCEND
      7E      04      AC      04      D4      0003C      BRB      1$
      FB8E      CF      CF      9F      00038      4$:      PUSHAB      CMP TABLE
      04      00048      7E      D4      0003C      CLRL      -(SP)
      03      FB      00043      ADDL3      #4, TABLE, -(SP)
      04      00048      CALLS      #3, SORT_LIST
      RET
```

; Routine Size: 73 bytes, Routine Base: \$CODE\$ + 04EF

```
: 1076      1166  1
: 1077      1167  1
: 1078      1168  1 ! Indices for getting to the process and directory tables via lnm$al_dirtbl.
: 1079      1169  1 !
: 1080      1170  1 literal
: 1081      1171  1      k_process_directory = 1,
: 1082      1172  1      k_system_directory = 0;
: 1083      1173  1
: 1084      1174  1 ! A little routine used by for_each_lnmth, below.
: 1085      1175  1 !
: 1086      1176  1 routine fe_lnmth(lnmth: ref $bblock, rtn, prm): novalue =
: 1087      1177  2      begin
: 1088      1178  2      local
: 1089      1179  2      p:      ref $bblock;
: 1090      1180  2      p = lnmth - %fieldexpand (lnmth$l_sibling,0);
: 1091      1181  2      while (p = .p[lnmth$l_sibling]) neq 0 do
: 1092      1182  3      begin
: 1093      1183  3      (.rtn)(p[base_], .prm);
```

```
: 1094      1184  3      fe_lnmth(.p[lnmth$_child], .rtn, .prm);
: 1095      1185  2      end;
: 1096      1186  1      end;
```

```
                                0004 00000 FE_LNMTH:
                                .WORD      Save R2
                                52      EF  AC  9E 00002      MOVAB      LNMTH-21, P
                                52      15  A2  D0 00006 1$:    MOVL      21(P), P
                                16      13  0000A      BEQL      2$
                                0C      AC  DD 0000C      PUSHL     PRM
                                52      DD 0000F      PUSHL     P
                                08      BC      02  FB 00011     CALLS     #2, @RTN
                                7E      08      AC  7D 00015     MOVQ      RTN, -(SP)
                                11      A2  DD 00019      PUSHL     17(P)
                                E0      AF      03  FB 0001C     CALLS     #3, FE_LNMTH
                                E4      11 00020      BRB       1$
                                04      00022 2$:    RET
                                : 1176
                                : 1180
                                : 1181
                                : 1183
                                : 1184
                                : 1181
                                : 1186
```

; Routine Size: 35 bytes, Routine Base: \$CODE\$ + 0538

```
: 1097      1187  1
: 1098      1188  1  ! This routine loops over all lnmth blocks,
: 1099      1189  1  ! and calls a supplied routine for each.
: 1100      1190  1  !
: 1101      1191  1 routine for_each_lnmth(rtn,prm): novalue =
: 1102      1192  2      begin
: 1103      1193  2      decr i from k_process_directory to k_system_directory do
: 1104      1194  3      begin
: 1105      1195  3      local
: 1106      1196  3      lnmth:      ref $bblock,
: 1107      1197  3      lnmx:      ref $bblock,
: 1108      1198  3      lnmb:      ref $bblock;
: 1109      1199  3      lnmb = ..lnm$al_dirtbl[i];
: 1110      1200  3      lnmx = ..lnmb[lnmb$_name] + 1+lnmb[lnmb$_name];
: 1111      1201  3      lnmth = 1+lnmx[lnmx$_xlation];
: 1112      1202  3      fe_lnmth(lnmth[base_], .rtn, .prm);
: 1113      1203  2      end;
: 1114      1204  1      end;
```

```
                                0004 00000 FOR_EACH_LNMTH:
                                .WORD      Save R2
                                52      01  D0 00002      MOVL      #1, I
                                50      00000000G0042  D0 00005 1$:    MOVL      LNM$AL_DIRTBL[I], R0
                                51      60  D0 0000D      MOVL      (R0), [NMB
                                50      11      A1  9A 00010     MOVZBL     17(LNMB), R0
                                50      12 A140  9E 00014     MOVAB      18(LNMB)[R0], LNMX
                                50      05  C0 00019      ADDL2     #5, LNMTH
                                7E      04  AC  7D 0001C     MOVQ      RTN, -(SP)
                                : 1191
                                : 1202
                                : 1199
                                : 1200
                                : 1201
                                : 1202
```


00000000G	00	E2	AF	9F	0001B	PUSHAB	FOR EACH LNMB	:
			02	FB	0001E	CALLS	#2, -SYS\$CMKRNL	:
		0000	CF	D4	00025	CLRL	SAVED_FP	:
				04	00029	RET		:
	54			66	D0 0002A	2\$:	MOVL	CTL\$GL_PCB, R4
		00000000G		00	16 0002D		JSB	LNMSLOCKR
FCD4	CF			00	FB 00033		CALLS	#0, RECHECK_TABLES
	55			01	D0 00038		MOVL	#1, I
	50	00000000G		45	D0 0003B	3\$:	MOVL	LNMSAL_HASHTBL[I], R0
	50			60	D0 00043		MOVL	(R0), [NMHSH
	52	0C		A0	9E 00046		MOVAB	12(R0), BUCKET
	53			60	D2 0004A		MCOML	(LNMHSH), R3
				53	D6 0004D		INCL	I
				14	11 0004F		BRB	6\$
	54			62	43 DE 00051	4\$:	MOVAL	(BUCKET)[I], LNMB
	54			64	D0 00055	5\$:	MOVL	(LNMB), LNMB
				0B	13 00058		BEQL	6\$
		08		AC	DD 0005A		PUSHL	PRM
				54	DD 0005D		PUSHL	LNMB
04	BC			02	FB 0005F		CALLS	#2, @RTN
				F0	11 00063		BRB	5\$
	E9			53	F4 00065	6\$:	SOBGEQ	I, 4\$
	D0			55	F4 00068		SOBGEQ	I, 3\$
	54			66	D0 0006B		MOVL	CTL\$GL_PCB, R4
		00000000G		00	16 0006E		JSB	LNMSUNLOCK
	12			00	DA 00074		MTPR	#0, #18
				04	00077		RET	

; Routine Size: 120 bytes, Routine Base: \$CODE\$ + 0585

- : 1261
- : 1263
- : 1264
- : .
- : 1265

; Routine Size: 16 bytes, Routine Base: \$CODES + 061F

```

1177 1266 1
1178 1267 1 !! Dump the quota for a logical name table
1179 1268 1 !!
1180 1269 1 routine dump_table_quot(table: ref table_block, args: ref vector): novalue =
1181 1270 2     begin
1182 1271 2
1183 1272 2         !! Don't dump the quota if this is a directory table,
1184 1273 2         !! or if there is no limit on the quota.
1185 1274 2
1186 1275 2         if (.table[table_b_flags] and lnmtb$m_directory) eql 0 and
1187 1276 2             .table[table_l_byteslm] neq 0
1188 1277 2         then
1189 1278 3             begin
1190 1279 3                 appstr_(' [Quota=(!!L,!!UL)]');
1191 1280 3                 apparg_(.table[table_l_bytes]);
1192 1281 3                 apparg_(.table[table_l_byteslm]);
1193 1282 2             end;
1194 1283 1         end;

```

```
.PSECT SPLITS, NOWRT, NOEXE, 2
```

```
2C 4C 55 21 28 3D 61 74 6F 75 51 58 20 20 13 000D4 P.AAW: .ASCII <19>\ [Quota=(!UL,!UL)]\
5D 29 4C 55 21 000E3
```

```
.PSECT $CODE$,NOWRT,2
```

				0004 00000 DUMP_TABLE QUOT:				
24	15	52	04	AC	D0 00002	WORD	Save R2	: 1269
		A2		01	E0 00006	MOVL	TABLE, R2	: 1275
			08	A2	D5 0000B	BBS	#1, 21(R2), 1\$	
				1F	13 0000E	TSTL	8(R2)	: 1276
			0000'	CF	9F 00010	BEQL	1\$	
			08	AC	DD C0014	PUSHAB	P.AAW	: 1279
		B3	AF	02	FB 00017	PUSHL	ARGS	
			0C	A2	DD 0001B	CALLS	#2, APPEND_STR	
			08	AC	DD 0001E	PUSHL	12(R2)	: 1280
		CB	AF	02	FB 00021	PUSHL	ARGS	
			08	A2	DD 00025	CALLS	#2, APPEND_ARG	
			08	AC	DD 00028	PUSHL	8(R2)	: 1281
C1	AF	02	FB 0002B	CALLS	#2, APPEND_ARG			
							04 0002F 1\$:	RET

```
; Routine Size: 48 bytes,   Routine Base: $CODE$ + 062F
```

```

: 1195      1284 1
: 1196      1285 1 : Dump logical name table attributes
: 1197      1286 1 :

```

```
1198 1287 1 routine dump_table_attr(table: ref table_block, args: ref vector): novalue =
1199 1288 2 begin
1200 1289 2 macro
1201 1290 2 f_(x) = (.table[table_b_flags] and %name('lnmth$m_',x)) neq 0 %;
1202 1291 2 local
1203 1292 2 cnt;
1204 1293 2
1205 1294 2 ! Save the current number of arguments
1206 1295 2
1207 1296 2 cnt = .args[0];
1208 1297 2
1209 1298 2 ! Append ascic strings for the various attributes
1210 1299 2
1211 1300 2 ! The following order was chosen, so that they form readable phrases.
1212 1301 2
1213 1302 2 if f_('shareable') then apparg_(ac_('shareable'));
1214 1303 2 if f_('group') then apparg_(ac_('group'));
1215 1304 2 if f_('system') then apparg_(ac_('system'));
1216 1305 2 if f_('directory') then apparg_(ac_('directory'));
1217 1306 2
1218 1307 2 ! Determine how many attributes we'll be showing (if any), and
1219 1308 2 ! append sufficient fao control strings for all of them.
1220 1309 2
1221 1310 2 if (cnt = .args[0] - .cnt) neq 0
1222 1311 2 then
1223 1312 2 begin
1224 1313 2 appstr_(' [!AC');
1225 1314 2 decr i from .cnt-1 to 1 do appstr_(',!AC');
1226 1315 2 appstr_(']');
1227 1316 2 end;
1228 1317 2 end;
```

.PSECT \$PLITS\$,NOWRT,NOEXE,2

65	6C	62	61	65	72	61	68	73	09	000E8	P.AAX:	.ASCII	<9>\shareable\	:
				70	75	6F	72	67	05	000F2	P.AAY:	.ASCII	<5>\group\	:
			6D	65	74	73	79	73	06	000F8	P.AAZ:	.ASCII	<6>\system\	:
79	72	6F	74	63	65	72	69	64	09	000FF	P.ABA:	.ASCII	<9>\directory\	:
			43	41	21	5B	20	20	06	00109	P.ABB:	.ASCII	<6>\ [!AC\	:
					43	41	21	2C	04	00110	P.ABC:	.ASCII	<4>\,!AC\	:
								5D	01	00115	P.ABD:	.ASCII	<1>\j\	:

.PSECT \$CODE\$,NOWRT,2

007C 00000 DUMP_TABLE_ATTR:

56	BB	AF	9E	00002	.WORD	Save R2,R3,R4,R5,R6	:	1287
55	0000	CF	9E	00006	MOVAB	APPEND_ARG, R6	:	
53	04	AC	7D	0000B	MOVAB	P.AAX, R5	:	1302
52		64	D0	0000F	MOVQ	TABLE, R3	:	1296
05	15	A3	E9	00012	MOVL	(R4), CNT	:	1302
		30	BB	00016	BLBC	21(R3), 1\$	:	
					PUSHR	#*M<R4,R5>	:	
66		02	FB	00018	CALLS	#2, APPEND_ARG	:	

08	15	A3	0A	02	E1	0001B	1\$:	BBC	#2, 21(R3), 2\$	1303
				A5	9F	00020		PUSHAB	P.AAY	
				54	DD	00023		PUSHL	R4	
08	15	66		02	FB	00025		CALLS	#2, APPEND_ARG	
		A3	10	03	E1	00028	2\$:	BBC	#3, 21(R3), 3\$	1304
				A5	9F	0002D		PUSHAB	P.AAZ	
				54	DD	00030		PUSHL	R4	
08	15	66		02	FB	00032		CALLS	#2, APPEND_ARG	
		A3	17	01	E1	00035	3\$:	BBC	#1, 21(R3), 4\$	1305
				A5	9F	0003A		PUSHAB	P.ABA	
				54	DD	0003D		PUSHL	R4	
		66		02	FB	0003F		CALLS	#2, APPEND_ARG	
52		64		52	C3	00042	4\$:	SUBL3	CNT, (R4), -CNT	1310
				1A	13	00046		BEQL	7\$	
			21	A5	9F	00048		PUSHAB	P.ABB	1313
				03	11	0004B		BRB	6\$	
			28	A5	9F	0004D	5\$:	PUSHAB	P.ABC	1314
				54	DD	00050	6\$:	PUSHL	R4	
	DE	A6		02	FB	00052		CALLS	#2, APPEND_STR	
		F4		52	F5	00056		SOBGTR	1, 5\$	
			2D	A5	9F	00059		PUSHAB	P.ABD	1315
				54	DD	0005C		PUSHL	R4	
	DE	A6		02	FB	0005E		CALLS	#2, APPEND_STR	
				04	00	0062	7\$:	RET		1317

; Routine Size: 99 bytes, Routine Base: \$CODE\$ + 065F

```
1229 1318 1
1230 1319 1
1231 1320 1 ! Return an ascic string for an access/integrity mode
1232 1321 1
1233 1322 1 routine ac_acmode(mode) =
1234 1323 2   begin
1235 1324 2     own
1236 1325 2     mode_str: vector[4] psect($plit$) preset(
1237 1326 2       [psl$c_kernel]= ac_('kernel'),
1238 1327 2       [psl$c_exec]=   ac_('exec'),
1239 1328 2       [psl$c_super]=   ac_('super'),
1240 1329 2       [psl$c_user]=    ac_('user'));
1241 1330 2   if .mode lssu 4
1242 1331 2   then
1243 1332 2     return .mode_str[.mode]
1244 1333 2   else
1245 1334 2     return ac_('?');
1246 1335 1   end;
```

.PSECT \$PLIT\$,NOWRT,NOEXF,2

6C	65	6E	72	65	6B	06	00117	P.ABE:	.ASCII	<6>\kernel\
		63	65	78	65	04	0011E	P.ABF:	.ASCII	<4>\exec\
	72	65	70	75	73	05	00123	P.ABC:	.ASCII	<5>\super\
		72	65	73	75	04	00129	P.ABH:	.ASCII	<4>\user\
							0012E		.BLKB	2

00000000' 00000000' 00000000' 00000000' 00130 MODE_STR:

3F 01 00140 P.ABI: .ADDRESS P.ABE, P.ABF, P.ABG, P.ABH
.ASCII <1>\?\

.PSECT \$CODE\$,NOWRT,2

0000 00000 AC_ACMODE:			
50	04	AC	D0 00002
04		50	D1 00006
		07	1E 00009
50	0000'	CF40	D0 0000B
		04	00011
50	0000'	CF	9E 00012 1\$:
		04	0001'

.WORD	Save nothing
MOVL	MODE, R0
CMPL	R0, #4
BGEQU	1\$
MOVL	MODE_STR[R0], R0
RET	
MOVAB	P.ABI, R0
RET	

: 1322
: 1330
: 1334
: 1335

; Routine Size: 24 bytes, Routine Base: \$CODE\$ + 06C2

: 1247	1336	1	
: 1248	1337	1	! Dump the access mode for a logical name table
: 1249	1338	1	!
: 1250	1339	1	routine_dump_table_mode(table: ref table_block, args: ref vector): novalue =
: 1251	1340	2	begin
: 1252	1341	2	! if .table[table_b_acmode] eql psl\$super then return;
: 1253	1342	2	appstr_(![!AC]);
: 1254	1343	2	apparg_(ac_acmode(.table[table_b_acmode]));
: 1255	1344	1	end;

.PSECT \$SPLIT\$,NOWRT,NOEXE,2

5D 43 41 21 5B 5F 21 07 00142 P.ABJ: .ASCII <7>\!_[!AC]\

.PSECT \$CODE\$,NOWRT,2

0000 00000 DUMP_TABLE_MODE:			
	0000'	CF	9F 00002
	08	AC	DD 00006
FF15	CF	02	FB 00009
	50	04	AC D0 0000E
	7E	14	A0 9A 00012
CE	AF	01	FB 00016
		50	DD 0001A
		08	AC DD 0001C
021	CF	02	FB 0001F
		04	00024

.WORD	Save nothing
PUSHAB	P.ABJ
PUSHL	ARGS
CALLS	#2, APPEND_STR
MOVL	TABLE, R0
MOVZBL	20(R0), -(SP)
CALLS	#1, AC_ACMODE
PUSHL	R0
PUSHL	ARGS
CALLS	#2, APPEND_ARG
RET	

: 1339
: 1342
: 1343
: 1344

; Routine Size: 37 bytes, Routine Base: \$CODE\$ + 06DA

: 1256 1345 1

```
1257 1346 1 ! Dump the attributes for a logical name
1258 1347 1 !
1259 1348 1 routine dump_log_attr(log: ref log_block, args: ref vector): novalue =
1260 1349 2 begin
1261 1350 2 macro
1262 1351 2 f_(x) = (.log[log_b_flags] and %name('lnmb$m_',x)) neq 0 %;
1263 1352 2 local
1264 1353 2 cnt;
1265 1354 2
1266 1355 2 cnt = .args[0];
1267 1356 2
1268 1357 2 if .log[log_acmode] neq psl$c_super
1269 1358 2 then
1270 1359 3 begin
1271 1360 3 apparg_(ac_acmode(.log[log_b_acmode]));
1272 1361 3 end;
1273 1362 2
1274 1363 2 ! The following are in alphabetical order, for lack of anything better.
1275 1364 2
1276 1365 2 if f_('confine') then apparg_(ac_('confine'));
1277 1366 2 if f_('crelog') then apparg_(ac_('crelog'));
1278 1367 2 if f_('no_alias') then apparg_(ac_('no_alias'));
1279 1368 2 if f_('nodelete') then apparg_(ac_('nodelete'));
1280 1369 2 if f_('table') then apparg_(ac_('table'));
1281 1370 2
1282 1371 2 if (cnt = .args[0] - .cnt) neq 0
1283 1372 2 then
1284 1373 3 begin
1285 1374 3 appstr_(' [!AC');
1286 1375 3 decr i from .cnt-1 to 1 do appstr_(',!AC');
1287 1376 3 appstr_(']');
1288 1377 2 end;
1289 1378 1 end;
```

.PSECT \$SPLITS,NOWRT,NOEXE,2

65	6E	69	66	6E	6F	63	07	0014A	P.ABK:	.ASCII	<7>\confine\	
	67	6F	6C	65	72	63	06	00152	P.ABL:	.ASCII	<6>\crelog\	
73	61	69	6C	61	5F	6F	08	00159	P.ABM:	.ASCII	<8>\no_alias\	
65	74	65	6C	65	64	6F	08	00162	P.ABN:	.ASCII	<8>\nodelete\	
			65	6C	62	61	74	05	00168	P.ABO:	.ASCII	<5>\table\
			43	41	21	5B	20	05	00171	P.ABP:	.ASCII	<5>\ [!AC\
				43	41	21	2C	04	00177	P.ABQ:	.ASCII	<4>\,!AC\
							5D	01	0017C	P.ABR:	.ASCII	<1>\]\

.PSECT \$CODE\$,NOWRT,2

007C 0000 DUMP_LOG_ATTR:

56	0000'	CF	9E	00002	WORD	Save R2,R3,R4,R5,R6	1348
55	FF15	CF	9E	00007	MOVAB	P.ABK, R6	
53	04	AC	7D	0000C	MOVAB	APPEND ARG, R5	1360
52		64	5D	00010	MOVQ	LOG, R3	
					MOVL	(R4), CNT	1355

	00A3	7E C5	09	A3 9A 00013 01 FB 00017 50 DD 0001C 54 DD 0001E	MOVZBL 9(R3), -(SP) CALLS #1, AC_ACMODE PUSHL R0 PUSHL R4		1360
07		65 53 53	08	02 FB 00020 A3 9A 00023 01 E1 00027	CALLS #2, APPEND_ARG MOVZBL 8(R3), R3 BBC #1, R3, 1\$		1365
08		65 53	0050	8F BB 0002B 02 FB 0002F 02 E1 00032 1\$:	PUSHR #*M<R4,R6> CALLS #2, APPEND_ARG BBC #2, R3, 2\$		1366
		65 08	08	A6 9F 00036 54 DD 00039 02 FB 0003B	PUSHAB P.ABI PUSHL R4 CALLS #2, APPEND_ARG		
		65 08	0F	53 E9 0003E 2\$: A6 9F 00041 54 DD 00044	BLBC R3, 3\$ PUSHAB P.ABM PUSHL R4		1367
08		65 53		02 FB 00046 04 E1 00049 3\$:	CALLS #2, APPEND_ARG BBC #4, R3, 4\$		1368
		65 53	18	A6 9F 0004D 54 DD 00050 02 FB 00052	PUSHAB P.ABN PUSHL R4 CALLS #2, APPEND_ARG		
08		65 53		03 E1 00055 4\$: A6 9F 00059 54 DD 0005C	BBC #3, R3, 5\$ PUSHAB P.ABO PUSHL R4		1369
52		65 64	21	02 FB 0005E 52 C3 00061 5\$: 1A 13 00065	CALLS #2, APPEND_ARG SUBL3 CNT, (R4), -CNT BEQL 8\$		1371
			27	A6 9F 00067 03 11 0006A	PUSHAB P.ABP BRB 7\$		1374
			2D	A6 9F 0006C 6\$: 54 DD 0006F 7\$: 02 FB 00071	PUSHAB P.ABQ PUSHL R4 CALLS #2, APPEND_STR		1375
	DE	A5 F4		52 F5 00075 A6 9F 00078	SOBGTR I, 6\$ PUSHAB P.ABR		1376
	DE	A5	32	54 DD 0007B 02 FB 0007D 04 00081 8\$:	PUSHL R4 CALLS #2, APPEND_STR RET		1378

. Routine Size: 130 bytes, Routine Base: \$CODE\$ + 06FF

```
. 1290      1379 1
. 1291      1380 1
. 1292      1381 1 ! Dump translation attributes
. 1293      1382 1 !
. 1294      1383 1 routine dump_lnm_attr(lmx: ref $block, args: ref vector): novalue =
. 1295      1384 2   begin
. 1296      1385 2   macro
. 1297      1386 2       f_(x) = .lnmx[%name('lnmx$V_',x)] i.
. 1298      1387 2   local
. 1299      1388 2       cnt;
. 1300      1389 2   cnt = .args[0];
. 1301      1390 2
. 1302      1391 2   ! The following are in alphabetical order, for lack of anything better.
. 1303      1392 2   !
. 1304      1393 2   if f_('concealed') then apparg_(ac_('concealed'));
. 1305      1394 2   if f_('terminal') then apparg_(ac_('terminal'));
```

```
: 1306      1395 2      if (cnt = .args[0] - .cnt) neq 0
: 1307      1396 2      then
: 1308      1397 2          begin
: 1309      1398 2              appstr_(['!AC']);
: 1310      1399 2              decr i from .cnt-1 to 1 do appstr_(['!AC']);
: 1311      1400 2              appstr_(['!']);
: 1312      1401 2          end;
: 1313      1402 1      end;
```

.PSECT \$SPLITS,NOWRT,NOEXE,2

```
64 65 6C 61 65 63 6E 6F 63 09 0017E P.ABS: .ASCII <9>\concealed\
6C 61 6E 69 6D 72 65 74 08 00188 P.ABT: .ASCII <8>\terminal\
43 41 21 5B 20 05 00191 P.ABU: .ASCII <5>\!AC\
43 41 21 2C 04 00197 P.ABV: .ASCII <4>\!AC\
5D 01 0019C P.ABW: .ASCII <1>\!\
```

.PSECT \$CODE\$,NOWRT,2

003C 00000 DUMP_LNMX ATTR:

```
55 FE98 CF 9E 00002 .WORD Save R2,R3,R4,R5 1383
54 0000 CF 9E 00007 MOVAB APPEND_ARG, R5
53 08 AC D0 0000C MOVAB P.ABS, R4
52 63 D0 00010 MOVL ARG, R3 1389
05 04 BC E9 00013 MOVL (R3), CNT
18 BB 00017 BLBC @LNMX, 1$ 1393
02 FB 00019 PUSHB #2, APPEND_ARG
08 04 BC 01 E1 0001C 1$: BBC #1, @LNMX, -2$ 1394
0A A4 9F 00021 PUSHAB P.ABT
53 DD 00024 PUSHAB R3
02 FB 00026 CALLS #2, APPEND_ARG
52 63 52 C3 00029 2$: SUBL3 CNT, (R3), -CNT 1395
1A 13 0002D BEQL 5$
13 A4 9F 0002F PUSHAB P.ABU 1398
03 11 00032 BRB 4$
19 A4 9F 00034 3$: PUSHAB P.ABV 1399
53 DD 00037 4$: PUSHAB R3
DE A5 02 FB 00039 CALLS #2, APPEND_STR
F4 52 F5 0003D SOBGTR 1, 3$
1E A4 9F 00040 PUSHAB P.ABW 1400
53 DD 00043 PUSHAB R3
DE A5 02 FB 00045 CALLS #2, APPEND_STR
04 00049 5$: RET 1402
```

: Routine Size: 74 bytes. Routine Base: \$CODE\$ + 0781

```
: 1314      1403 1
: 1315      1404 1
: 1316      1405 1 : Dump translation strings and their attributes
: 1317      1406 1
: 1318      1407 1 routine dump_log_lnm(x(log: ref log_block, args: ref vector): novalue =
```

```
1319 1408 2 begin
1320 1409 2 local
1321 1410 2 first: initial(true),
1322 1411 2 lnmx: ref $bblock;
1323 1412 2 builtin
1324 1413 2 ap
1325 1414 2 callg;
1326 1415 2 lnmx = .log[log_t_name] + 1+log[log_t_name];
1327 1416 2 while not .lnmx[lnmx$V_xend] do
1328 1417 2 begin
1329 1418 2
1330 1419 2 Don't dump logical names that have "funny" indices.
1331 1420 2
1332 1421 2 if .lnmx[lnmx$b_index] geq 0 or .lnmx[lnmx$b_index] eql lnmx$C_table
1333 1422 2 then
1334 1423 2 begin
1335 1424 2 local len,adr;
1336 1425 2 if .first
1337 1426 2 then appstr_(' = '!AF'')
1338 1427 2 else appstr_('! = '!AF'');
1339 1428 2 len = .lnmx[lnmx$t_xlation];
1340 1429 2 adr = 1+lnmx[lnmx$t_xlation];
1341 1430 2 if .lnmx[lnmx$b_index] eql lnmx$C_table
1342 1431 2 then
1343 1432 2 len = 0;
1344 1433 2 if .len gequ 4 then if .(.adr)<0,16,0> eql %x'001b'
1345 1434 2 then
1346 1435 2 (len = .len - 4; adr = .adr + 4);
1347 1436 2 apparg_(.len);
1348 1437 2 apparg_(.adr);
1349 1438 2 if .flags[flag_v_full]
1350 1439 2 then
1351 1440 2 dump lnmx_attr(lnmx[base_], args[0]);
1352 1441 2 if .first and .flags[flag_v_recurse]
1353 1442 2 then
1354 1443 2 begin
1355 1444 2 bind
1356 1445 2 table = .log[log_l_table]: table_block;
1357 1446 2 appstr_(' (!AC)');
1358 1447 2 apparg_(table[table_t_name]);
1359 1448 2 end;
1360 1449 2 callg(args[0], util_output);
1361 1450 2
1362 1451 2 ! Reinitialize the argument list
1363 1452 2
1364 1453 2 begin
1365 1454 2 bind desc = .args[1]: vector[2];
1366 1455 2 args[0] = 0;
1367 1456 2 apparg_(desc[0]);
1368 1457 2 desc[0] = 0;
1369 1458 2 end;
1370 1459 2
1371 1460 2 first = false; ! No longer the first time here
1372 1461 2 end;
1373 1462 2 lnmx = .lnmx[lnmx$t_xlation] + 1+lnmx[lnmx$t_xlation];
1374 1463 2 end;
1375 1464 2 if not .first then flags[flag_v_found] = true;
```

; 1376

1465 1 end;

```
.PSECT $PLI$,NOWRT,NOEXE,2
22 22 46 41 21 22 20 3D 20 08 0019E P.ABX: .ASCII <8>\ = '!AF'\
46 46 41 21 22 20 3D 5F 21 09 001A7 P.ABY: .ASCII <9>\! = '!AF'\
29 43 41 21 28 20 06 001B1 P.ABZ: .ASCII <6>\ T!AC)\

.PSECT $CODE$,NOWRT,2
00FC 00000 DUMP_LOG_LNMX:
57 FE4E CF 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7 : 1407
56 01 D0 00007 MOVAB APPEND_ARG, R7 : 1408
51 04 AC D0 0000A MOVL #1, FIRST : 1415
50 0A A1 9A 0000E MOVZBL LOG, R1
52 0B A140 9E 00012 MOVAB 10(R1), R0
62 02 E1 00017 1$: BBC #2 (LNMX), 2$ : 1416
00A6 31 0001B BRW 11$
54 01 A2 98 0001E 2$: CVTBL 1(LNMX), R4 : 1421
09 18 00022 BGEQ 3$
82 8F 54 91 00024 CMPB R4, #-126
03 03 13 00028 BEQL 3$
008B 31 0002A BRW 10$
55 08 AC D0 0002D 3$: MOVL ARGS, R5 : 1426
06 56 E9 00031 BLBC FIRST, 4$ : 1425
0000' CF 9F 00034 PUSHAB P.ABX : 1426
04 11 00038 BRB 5$
0000' CF 9F 0003A 4$: PUSHAB P.ABY : 1427
55 DD 0003E 5$: PUSHL R5
DE A7 02 FB 00040 CALLS #2, APPEND_STR
50 04 A2 9A 00044 MOVZBL 4(LNMX), LEN : 1428
53 05 A2 9E 00048 MOVAB 5(R2), ADR : 1429
82 8F 54 91 0004C CMPB R4, #-126 : 1430
02 12 00050 BNEQ 6$
50 D4 00052 CLRL LEN : 1432
04 50 D1 00054 6$: CMPL LEN, #4 : 1433
0B 1F 00057 BLSSU 7$
1B 63 B1 00059 CMPW (ADR), #27
06 12 0005C BNEQ 7$
50 04 C2 0005E SUBL2 #4, LEN : 1435
53 04 C0 00061 ADDL2 #4, ADR
50 DD 00064 7$: PUSHL LEN : 1436
55 DD 00066 PUSHL R5
67 02 FB 00068 CALLS #2, APPEND_ARG
53 DD 0006B PUSHL ADR : 1437
55 DD 0006D PUSHL R5
07 0000' 67 02 FB 0006F CALLS #2, APPEND_ARG
CF 06 E1 00072 BBC #6, FLAGS, -8$ : 1438
24 BB 00078 PUSHR #*M<R2,R5> : 1440
0162 C7 02 FB 0007A CALLS #2, DUMP_LNMX_ATTR
1E 56 E9 0007F 8$: BLBC FIRST, 9$ : 1441
0000' CF 95 00082 TSTB FLAGS
```

			18	18	00086		JEQ	9\$		
	53	04	AC	D0	00088		MOVL	LOG, R3		1445
		0000	CF	9F	0008C		PUSHAB	P.ABZ		1446
			55	DD	00090		PUSHL	R5		
7E	DE	A7	02	FB	00092		CALLS	#2, APPEND_STR		
	04	A3	16	C1	00096		ADDL3	#22, 4(R3); -(SP)		1447
			55	DD	00098		PUSHL	R5		
		67	02	FB	0009D		CALLS	#2, APPEND_ARG		
	F9E1	C7	65	FA	000A0	9\$:	CALLG	(R5), UTIL_OUTPUT		1449
		50	08	AC	D0	000A5	MOVL	ARGS, R0		1454
		53	04	A0	D0	000A9	MOVL	4(R0), R3		
			60	D4	000AD		CLRL	(R0)		1455
			09	BB	000AF		PUSHR	#*M<R0,R3>		1456
		67	02	FB	000B1		CALLS	#2, APPEND_ARG		
			63	D4	000B4		CLRL	(R3)		1457
			56	D4	000B6		CLRL	FIRST		1460
		50	04	A2	9A	000B8	10\$:	MOVZBL	4(LNMX), R0	1462
		52	05	A240	9E	000BC	MOVAB	5(LNMX)[R0], LNMX		
			FF53	31	000C1		BRW	1\$		1416
		05	56	E8	000C4	11\$:	BLBS	FIRST, 12\$		1464
	0000	CF	01	88	000C7		BISB2	#1, FLAGS+1		
			04	000CC	12\$:	RET				1465

; Routine Size: 205 bytes, Routine Base: \$CODE\$ + 07CB

```
1377 1466 1
1378 1467 1 ! Dump a logical name and its attributes, and translations and their attributes
1379 1468 1
1380 1469 1 routine dump_log(
1381 1470 1     log: ref log_block,
1382 1471 1     depth
1383 1472 1 ): novalue =
1384 1473 2     begin
1385 1474 2     local
1386 1475 2         buf: vector[80,byte],
1387 1476 2         desc: vector[2],
1388 1477 2         args: vector[20];
1389 1478 2     macro
1390 1479 2         f(x) = (.log[log_b_flags] and %name('lnmb$m_',x)) neq 0 %;
1391 1480 2     builtin
1392 1481 2         actualcount,
1393 1482 2         testbitsc,
1394 1483 2         callg;
1395 1484 2     args[0] = 0;
1396 1485 2     apparg(desc[0]);
1397 1486 2     desc[0] = 0;
1398 1487 2     desc[1] = buf[0];
1399 1488 2     if actualcount() gequ 2
1400 1489 2     then
1401 1490 3         begin
1402 1491 3             apparg(.depth);
1403 1492 3             if .depth gtr 0
1404 1493 3                 then appstr('!UL  'AC'')
1405 1494 3                 else appstr('!+  'AC'');
1406 1495 3         end
1407 1496 2     else
```

```
1408      appstr(' '!AC'');
1409      apparg(log[log_t_name]);
1410      if .flags[flag_v_full]
1411      then
1412      begin
1413          Dump all of the attributes
1414          dump_log_attr(log[base_], args[0]);
1415      end
1416      else if f_('table')
1417      then
1418      begin
1419          Let the user know that this is a table
1420          appstr(' [table]');
1421      end
1422      else if .log[log_v_equal]
1423      then
1424      begin
1425          Display the access mode, so the logicals are distinguishable
1426          apparg(ac_acmode(.log[log_b_acmode]));
1427          appstr(' [.AC]');
1428      end
1429      dump_log_lnm(x(log[base_], args[0]);
1430      end;
```

```
                .PSECT $SPLITS$,NOWRT,NOEXE,2
22 43 41 21 22 20 20 4C 55 21 0A 001B8 P.ACA: .ASCII <10>\!UL '!AC'\
22 43 41 21 22 20 20 20 2B 21 0A 001C3 P.ACB: .ASCII <10>\!+ '!AC'\
      22 43 41 21 22 20 20 07 001CE P.ACC: .ASCII <7>\ '!AC'\
      5D 65 6C 62 61 74 5B 20 08 001D6 P.ACD: .ASCII <8>\ [table]\
      5D 43 41 21 5B 20 06 001DF P.ACE: .ASCII <6>\ [.AC]\
```

```
                .PSECT $CODE$,NOWRT,2
001C 00000 DUMP_LOG:
      54 0000' CF 9E 00002 .WORD Save R2,R3,R4
      53 FD7C CF 9E 00007 MOVAB P.ACA, R4
      5E FF5C CE 9E 0000C MOVAB APPEND_ARG, R3
      7E D4 00011 CLRL -164(SP), SP
      50 AE 9F 00013 PUSHAB ARGS
      04 AE 9F 00016 PUSHAB DESC
      63 02 FB 00019 CALLS #2, APPEND_ARG
      50 AE D4 0001C CLRL DESC
      54 AE 5B AE 9E 0001F MOVAB BUF, DESC+4
      02 6C 91 00024 CMPB (AP), #2
      17 1F 00027 BLSSU 2$
```

```
1469
1484
1485
1486
1487
1488
```

			08	AC	DD	00029	PUSHL	DEPTH	:	1491
			04	AE	9F	0002C	PUSHAB	ARGS	:	
		63		02	FB	0002F	CALLS	#2, APPEND_ARG	:	
			08	AC	D5	00032	TSTL	DEPTH	:	1492
				04	15	00035	BLEQ	1\$	:	
				54	DD	00037	PUSHL	R4	:	1493
				08	11	00039	BRB	3\$	:	
			08	A4	9F	0003B	1\$: PUSHAB	P.ACB	:	1494
				03	11	0003E	BRB	3\$	:	
			16	A4	9F	00040	2\$: PUSHAB	P.ACC	:	1497
			04	AE	9F	00043	3\$: PUSHAB	ARGS	:	
	DE	A3		02	FB	00046	CALLS	#2, APPEND_STR	:	
		52		04	AC	D0	MOVL	LOG, R2	:	1498
			0A	A2	9F	0004E	PUSHAB	10(R2)	:	
			04	AE	9F	00051	PUSHAB	ARGS	:	
		63		02	FB	00054	CALLS	#2, APPEND_ARG	:	
0B	0000'	CF		06	E1	00057	BBC	#6, FLAGS, -4\$	:	1499
			4004	8F	BB	0005D	PUSHR	#*M<R2,SP>	:	1505
	00E0	C3		02	FB	00061	CALLS	#2, DUMP_LOG_ATTR	:	
				2A	11	00066	BRB	7\$	:	1499
05	08	A2		03	E1	00068	4\$: BBC	#3, 8(R2), 5\$	:	1507
			1E	A4	9F	0006D	PUSHAB	P.ACD	:	1513
				19	11	00070	BRB	6\$	:	
			08	A2	95	00072	5\$: TSTB	8(R2)	:	1515
				1B	18	00075	BGEQ	7\$	:	
		7E		09	A2	00077	MOVZBL	9(R2), -(SP)	:	1521
	00A3	C3		01	FB	0007B	CALLS	#1, AC_ACMODE	:	
				50	DD	00080	PUSHL	R0	:	
			04	AE	9F	00082	PUSHAB	ARGS	:	
		63		02	FB	00085	CALLS	#2, APPEND_ARG	:	
			27	A4	9F	00088	PUSHAB	P.ACE	:	1522
			04	AE	9F	0008B	6\$: PUSHAB	ARGS	:	
	DE	A3		02	FB	0008E	CALLS	#2, APPEND_STR	:	
			4004	8F	BB	00092	7\$: PUSHR	#*M<R2,SP>	:	1524
	0'AC	C3		02	FB	00096	CALLS	#2, DUMP_LOG_LNMX	:	
				04	0009B		RET		:	1525

; Routine Size: 156 bytes, Routine Base: \$CODE\$ + 0898

```
1437 1526 1
1438 1527 1 ! Dump a logical name table and its attributes, and the contained logical names
1439 1528 1 !
1440 1529 1 routine dump_table(
1441 1530 1     table: ref table_block,
1442 1531 1     depth
1443 1532 1 ): novalue =
1444 1533 2     begin
1445 1534 2     local
1446 1535 2         buf: vector[80,byte],
1447 1536 2         desc: vector[2],
1448 1537 2         args: vector[20];
1449 1538 2     builtin
1450 1539 2         callg;
1451 1540 2
1452 1541 2     ! Note that we write the name of the table before checking whether there
1453 1542 2     ! are actually any logical names to be displayed in that table.
```

```
1454 1543 2 ! The idea for this is that it provides useful feedback about which tables
1455 1544 2 ! were searched.
1456 1545 2 ! This can be changed by including:
1457 1546 2
1458 1547 2 !!!!if .flags[flag_v_normal] and .table[table_l_log] eql 0 then return;
1459 1548 2
1460 1549 2 ! Display the table name and attributes
1461 1550 2
1462 1551 2 if .flags[flag_v_normal] then write_blank_line();
1463 1552 2 args[0] = 0;
1464 1553 2 apparg(desc[0]);
1465 1554 2 desc[0] = 0;
1466 1555 2 desc[1] = buf[0];
1467 1556 2 if not .flags[flag_v_normal] then appstr('!#* ');
1468 1557 2 appstr('(!AC) ');
1469 1558 2 if not .flags[flag_v_normal] then apparg(4*min(.depth,10));
1470 1559 2 apparg(table[table_f_name]);
1471 1560 2
1472 1561 2 ! Dump attributes, if needed
1473 1562 2
1474 1563 2 if .flags[flag_v_full]
1475 1564 2 then
1476 1565 2 begin
1477 1566 2 dump_table_mode(table[base_], args[0]);
1478 1567 2 dump_table_attr(table[base_], args[0]);
1479 1568 2 dump_table_quot(table[base_], args[0]);
1480 1569 2 end;
1481 1570 2 callg(args[0], util_output);
1482 1571 2
1483 1572 2 ! If there are logical names in this table, display them.
1484 1573 2
1485 1574 2 if .table[table_l_log] neq 0
1486 1575 2 then
1487 1576 2 begin
1488 1577 2 local
1489 1578 2 log: ref log_block;
1490 1579 2 write_blank_line();
1491 1580 2 sort_list(table[table_l_log], %fieldexpand(log_l_next,0), cmp_log );
1492 1581 2 log = table[table_l_log] - %fieldexpand(log_l_next,0);
1493 1582 2 while (log = .log[log_l_next]) neq 0 do
1494 1583 2 dump_log(log[base_]);
1495 1584 2 end;
1496 1585 1 end;
```

.PSECT \$SPLITS,NOWRT,NOEXE,2

```
29 20 2A 23 21 04 001E6 P.ACF: .ASCII <4>\!#* \
29 43 41 21 28 05 001EB P.ACG: .ASCII <5>\(!AC)\
```

.PSECT \$CODE\$,NOWRT,2

001C 00000 DUMP_TABLE:
.WORD Save R2,R3,R4

: 1529

07	54	0000'	CF	9E	00002	MOVAB	FLAGS, R4	:	
	53	F6C1	CF	9E	00007	MOVAB	UTIL_OUTPUT, R3	:	
	5E	FF58	CE	9E	0000C	MOVAB	-168(SP), SP	:	
	64		04	E1	00011	BBC	#4, FLAGS, 1\$	:	1551
		0000'	CF	9F	00015	PUSHAB	BLANK LINE	:	
	63		01	FB	00019	CALLS	#1, UTIL_OUTPUT	:	
			6E	D4	0001C	CLRL	ARGS	:	1552
		50	AE	9F	0001E	PUSHAB	DESC	:	1553
		04	AE	9F	00021	PUSHAB	ARGS	:	
	061F	C3	02	FB	00024	CALLS	#2, APPEND_ARG	:	
			50	AE	00029	CLRL	DESC	:	1554
	54	AE	58	AE	9E	0002C	MOVAB	BUF, DESC+4	1555
0C	64		04	E0	00031	BBS	#4, FLAGS, 2\$	:	1556
		0000'	CF	9F	00035	PUSHAB	P.ACF	:	
		04	AE	9F	00039	PUSHAB	ARGS	:	
	05FD	C3	02	FB	0003C	CALLS	#2, APPEND_STR	:	
		0000'	CF	9F	00041	PUSHAB	P.ACG	:	1557
		04	AE	9F	00045	PUSHAB	ARGS	:	
	05FD	C3	02	FB	00048	CALLS	#2, APPEND_STR	:	
18	64		04	E0	0004D	BBS	#4, FLAGS, -4\$	:	1558
	50	08	AC	D0	00051	MOVL	DEPTH, R0	:	
	0A		50	D1	00055	CMPL	R0, #10	:	
			03	15	00058	BLEQ	3\$	:	
	50		0A	D0	0005A	MOVL	#10, R0	:	
7E	50		02	78	0005D	ASHL	#2, R0, -(SP)	:	
		04	AE	9F	00061	PUSHAB	ARGS	:	
	061F	C3	02	FB	00064	CALLS	#2, APPEND_ARG	:	
7E	04	AC	16	C1	00069	ADDL3	#22, TABLE, -(SP)	:	1559
		04	AE	9F	0006E	PUSHAB	ARGS	:	
	061F	C3	02	FB	00071	CALLS	#2, APPEND_ARG	:	
1E	64		06	E1	00076	BBC	#6, FLAGS, -5\$	:	1563
			5E	DD	0007A	PUSHL	SP	:	1566
		04	AC	DD	0007C	PUSHL	TABLE	:	
	06DA	C3	02	FB	0007F	CALLS	#2, DUMP_TABLE_MODE	:	
			5E	DD	00084	PUSHL	SP	:	1567
	065F	C3	04	AC	DD	00086	PUSHL	TABLE	
			02	FB	00089	CALLS	#2, DUMP_TABLE_ATTR	:	1568
		04	5E	DD	0008E	PUSHL	SP	:	
	062F	C3	02	FB	00093	PUSHL	TABLE	:	
	63		02	FB	00093	CALLS	#2, DUMP TABLE QUOT	:	1570
	50	04	6E	FA	00098	CALLG	ARGS, UTIL_OUTPUT	:	1574
	52	10	AC	D0	0009B	MOVL	TABLE, R0	:	
			A0	9E	0009F	MOVAB	16(R0), R2	:	
			62	D5	000A3	TSTL	(R2)	:	
			22	13	000A5	BEQL	7\$	:	
		0000'	CF	9F	000A7	PUSHAB	BLANK LINE	:	1579
	63		01	FB	000AB	CALLS	#1, UTIL_OUTPUT	:	
		03E7	C3	9F	000AE	PUSHAB	CMP LOG	:	1580
			7E	D4	000B2	CLRL	-(SP)	:	
			52	DD	000B4	PUSHL	R2	:	
	00C5	C3	03	FB	000B6	CALLS	#3, SORT_LIST	:	
	52		62	D0	000BB	MOVL	(LOG), LOG	:	1582
			09	13	000BE	BEQL	7\$	:	
			52	DD	000C0	PUSHL	LOG	:	1583
	0898	C3	01	FB	000C2	CALLS	#1, DUMP_LOG	:	
			F2	11	000C7	BRB	6\$	:	
			04	000C9	7\$:	RET		:	1585

SHOWLOG
V04-000

³
16-Sep-1984 01:28:17
14-Sep-1984 12:09:37

VAX-11 Bliss-32 V4.0-742
[CLIUTL.SRC]SHOWLOG.B32;1

Page 54
(10)

; Routine Size: 202 bytes, Routine Base: \$CODE\$ + 0934

```
1498 1586 1 ! Find all the relevant logical name tables specified by the user.
1499 1587 1 ! This routine uses cmdtbl_list, and calls add_to_known to create table_blocks.
1500 1588 1
1501 1589 1 routine get_relevant_tables =
1502 1590 2 begin
1503 1591 2 local
1504 1592 2 list: ref cmdtbl_block;
1505 1593 2
1506 1594 2 runs_in_kernel_(get_relevant_tables);
1507 1595 2
1508 1596 2 ! Traverse the list, and process each name as either:
1509 1597 2 ! A wildcard string (find all table names that match)
1510 1598 2 ! A table name (process that table)
1511 1599 2 ! A search list (process all relevant tables)
1512 1600 2
1513 1601 2 list = cmdtbl_list - %fieldexpand(cmdtbl_l_next,0);
1514 1602 2 while (list = .list[cmdtbl_l_next]) neq 0 do
1515 1603 3 begin
1516 1604 3 bind name = list[cmdtbl_q_name]: $bblock[dsc$d_bln];
1517 1605 3 if not ch$fail(ch$find_ch(.name[dsc$w_length],.name[dsc$a_pointer], '*'))
1518 1606 3 or not ch$fail(ch$find_ch(.name[dsc$w_length],.name[dsc$a_pointer], '%'))
1519 1607 3 then
1520 1608 4 begin
1521 1609 4 ! A wildcard string (find all table names that match)
1522 1610 4
1523 1611 4 for each lnmth( match lnmth, list[base] );
1524 1612 4 sort_list( list[cmdtbl_l_table], %fieldexpand(table_l_next,0),
1525 1613 4 cmp_table );
1526 1614 4
1527 1615 4 end
1528 1616 3 else
1529 1617 4 begin
1530 1618 4 ! A table name (process that table)
1531 1619 4 ! A search list (process all relevant tables)
1532 1620 4
1533 1621 4 ! In either case, we simply use lnm$setup and lnm$table.
1534 1622 4
1535 1623 4 ??? Note that there is a slight problem with the access mode.
1536 1624 4 If we set it to .acmode, then SHOW LOGICAL /EXEC causes no tables
1537 1625 4 to be found, since LNM$DCL_LOGICAL is a supervisor mode name.
1538 1626 4 Instead, we set it to psl$user, and let add_to_known check the
1539 1627 4 access mode of the resulting table.
1540 1628 4
1541 1629 4 local
1542 1630 4 an_rt_block: rt_block,
1543 1631 4 lnmth: ref $bblock,
1544 1632 4 status;
1545 1633 4
1546 1634 4 register
1547 1635 4 rt = 5: ref rt_block initial(an_rt_block[base]);
1548 1636 4 rt[rt_b_acmode] = psl$user; ! Acceptable integrity mode
1549 1637 4 rt[rt_b_flags] = 0; ! Initialize flags
1550 1638 4 rt[rt_v_case] = true; ! Case-insensitive
1551 1639 4 status = lnm$setup(.name[dsc$w_length],.name[dsc$a_pointer],
1552 1640 4 rt[base]; lnmth);
1553 1641 4 while true do
1554 1642 5 begin
```

```
1555      1643 5      if not .status
1556      1644 5      then
1557      1645 6          begin
1558      1646 6              if .status eql ss$_toomanylnam
1559      1647 6                  then
1560      1648 6                      rt[rt_b_depth] = .rt[rt_b_depth] - 1      ! Backup depth
1561      1649 6                  else
1562      1650 6                      exitloop      ! Presumably no more names
1563      1651 6                  end
1564      1652 5              else
1565      1653 6                  begin
1566      1654 6                      local table: ref table_block;
1567      1655 6                      if (table = add_to_known(lnmth[base_])) neq 0
1568      1656 6                          then
1569      1657 6                              util_slide(table[table_l_next],
1570      1658 6                                  list[cmdtbl_l_table], table[base_]);
1571      1659 5                      end;
1572      1660 5                      rt[rt_b_tries] = rt_c_maxtries; ! Reinitialize number of tries
1573      1661 5                      status = lnm$stable(rt[base_]; lnmth);
1574      1662 4                      end;
1575      1663 4                      reverse_list(list[cmdtbl_l_table], %fieldexpand_(table_l_next,0));
1576      1664 3                      end;
1577      1665 2      end;
1578      1666 2
1579      1667 2      ! Reverse the order of known_list, so that traversing the list will
1580      1668 2      ! visit the tables in the same order as they were found.
1581      1669 2      reverse_list(known_list, %fieldexpand_(known_l_next,0));
1582      1670 2
1583      1671 2      ! Now get the descendants, if requested.
1584      1672 2
1585      1673 2      ! Note that this is done after finding all the tables specified on the
1586      1674 2      ! command line. Thus, the user's requested order is honored as much as
1587      1675 2      ! possible. It also avoids anomolous behaviour for '/table=*/descendants'.
1588      1676 2
1589      1677 2      ??? However, it gives odd behaviour for the command
1590      1678 2      ! '/table=lnm$* directory/descendants/process/group/job/system'
1591      1679 2      ! as compared with saying '/table=*dire*/descendants'.
1592      1680 2
1593      1681 2      if .flags[flag_v_descend]
1594      1682 2      then
1595      1683 2          begin
1596      1684 2              ! We traverse the known list in the order in which things were found
1597      1685 2              local
1598      1686 2              known: ref known_block;
1599      1687 2              known = known_list - %fieldexpand_(known_l_next,0);
1600      1688 2              while (known = .known[known_l_next]) neq 0 do
1601      1689 2                  begin
1602      1690 2                      local
1603      1691 2                      new: ref known_block;
1604      1692 2                      get_descend(.known[known_l_table], .known[known_l_lnmth]);
1605      1693 2                      new = known_list - %fieldexpand_(known_l_next,0);
1606      1694 2                      while .new[known_l_next] neq known[base_] do
1607      1695 2                          new = .new[known_l_next];
1608      1696 2                      if known_list[base_] neq known[base_]      ! Any new tables?
```

```

: 1612      1700      4
: 1613      1701      3
: 1614      1702      3
: 1615      1703      3
: 1616      1704      3
: 1617      1705      4
: 1618      1706      3
: 1619      1707      2
: 1620      1708      2
: 1621      1709      2
: 1622      1710      2
: 1623      1711      2
: 1624      1712      2
: 1625      1713      1

      then
      begin
        known_list = util_slide(new[known_l_next], known[known_l_next],
        known_list[base_]);
        known = new[base_];
      end;
    end;
  end;

  lnmsunlock(.ctl$gl_pcb);
  set_ipl(0);

  return ss$_normal;
end;
```

```

                                07FC 00000 GET_RELEVANT TABLES:
                                .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10
                                MOVAB CTL$GL_PCB, R10
                                MOVAB KNOWN_LIST, R9
                                MOVAB UTIL_SLIDE, R8
                                SUBL2 #48, -SP
                                MOVPSL PSL
                                CMPZV #24, #2, PSL, #0
                                BNEQ 1$
                                MOVL FP, SAVED_FP
                                BRB 2$
                                PUSHL AP
                                PUSHAB GET_RELEVANT TABLES
                                CALLS #2, -SYS$CMKRNL
                                CLRL SAVED_FP
                                RET
                                MOVL CTL$GL_PCB, R4
                                JSB LNMSLOCKR
                                MOVAB CMDTBL_LIST, LIST
                                MOVL (LIST), LIST
                                BNEQ 4$
                                BRW 14$
                                MOVAB 8(LIST), R6
                                MOVAB 4(R4), R7
                                LOCC #42, (R6), @4(R6)
                                BNEQ 5$
                                CLRL R1
                                TSTL R1
                                BNEQ 7$
                                LOCC #37, (R6), @4(R6)
                                BNEQ 6$
                                CLRL R1
                                TSTL R1
                                BEQL 9$
                                PUSHL LIST
                                PUSHAB MATCH_LNMTN
                                CALLS #2, FOR_EACH_LNMTN
                                PUSHAB CMP_TABLE

00          50          02          18 ED 00018          1$
          FC A9          06 12 0001D          2$
          5C DD 00025          3$
          099E C8 9F 00027          4$
          FC A9 D4 00032          5$
          54 00000000G 6A D0 00036          6$
          54 04 A9 9E 0003F          7$
          54 64 D0 00043          8$
          03 12 00046          9$
          0094 31 00048          10$
          56 08 A4 9E 0004B          11$
          57 04 A4 9E 0004F          12$
          04 B6 66 2A 3A 00053          13$
          02 12 00058          14$
          51 D4 0005A          15$
          51 D5 0005C          16$
          0D 12 0005E          17$
          25 3A 00060          18$
          02 12 00065          19$
          51 D4 00067          20$
          51 D5 00069          21$
          19 13 0006B          22$
          54 DD 0006D          23$
          04FB C8 0326 C8 9F 0006F          24$
          0367 C8 FB 00073          25$
          C8 9F 00078          26$
```

			7E	D4	0007C	CLRL	-(SP)		
			57	DD	0007E	PUSHL	R7		
65	A8		03	FB	00080	CALLS	#3, SORT_LIST		
			BD	11	00084	BRB	3\$		1605
	55		6E	9E	00086	MOVAB	AN_RT_BLOCK, RT		1635
	65		03	80	00089	MOVW	#3, (RT)		1636
01	A5		01	88	0008C	BISB2	#1, 1(RT)		1638
	53	04	A6	D0	00090	MOVL	4(R6), R3		1640
	52		66	3C	00094	MOVZWL	(R6), R2		
		00000000G	0C	16	00097	JSB	LNMS\$SETUP		
	52		50	D0	0009D	MOVL	R0, STATUS		
	56		51	D0	000A0	MOVL	R1, R6		
	0E		52	E8	000A3	BLBS	STATUS, 11\$		1643
00000374	8F		52	D1	000A6	CHPL	STATUS, #884		1646
			26	12	000AD	BNEQ	13\$		
		02	A5	97	000AF	DECB	2(RT)		1648
			15	11	000B2	BRB	12\$		
			56	DD	000B4	PUSHL	LNMTM		1655
02B4	C8		01	FB	000B6	CALLS	#1, ADD_TO_KNOWN		
			50	D5	000BB	TSTL	TABLE		
			CA	13	000BD	BEQL	12\$		
		04	50	DD	00CBF	PUSHL	TABLE		1658
			A4	9F	000C1	PUSHAB	4(LIST)		
			50	DD	000C4	PUSHL	TABLE		
	68		03	FB	000C6	CALLS	#3, UTIL_SLIDE		
03	A5		01	8E	000C9	MNEGB	#1, 3(RT)		1660
		00000000G	00	16	000CD	JSB	LNMT\$TABLE		1661
			C8	11	000D3	BRB	10\$		
			7E	D4	000D5	CLRL	-(SP)		1663
			57	DD	000D7	PUSHL	R7		
46	A8		02	FB	000D9	CALLS	#2, REVERSE_LIST		
			A5	11	000DD	BRB	8\$		1602
			7E	D4	000DF	CLRL	-(SP)		1670
			59	DD	000E1	PUSHL	R9		
	46		02	FB	000E3	CALLS	#2, REVERSE_LIST		
36	0C		05	E1	000E7	BBC	#5, FLAGS, T8\$		1682
			69	9E	000EC	MOVAB	KNOWN_LIST, KNOWN		1690
			62	D0	000EF	MOVL	(KNOWN), KNOWN		1691
			2E	13	000F2	BEQL	18\$		
		04	A2	DD	000F4	PUSHL	4(KNOWN)		1695
		08	A2	DD	000F7	PUSHL	8(KNOWN)		
048F	C8		02	FB	000FA	CALLS	#2, GET_DESCEND		
	53		69	9E	000FF	MOVAB	KNOWN_LIST, NEW		1696
	52		63	D1	00102	CMPL	(NEW), KNOWN		1697
			05	13	00105	BEQL	17\$		
	53		63	D0	00107	MOVL	(NEW), NEW		1698
			F6	11	0010A	BRB	16\$		
	52		69	D1	0010C	CMPL	KNOWN_LIST, KNOWN		1699
			DE	13	0010F	BEQL	15\$		
			69	DD	00111	PUSHL	KNOWN_LIST		1703
			52	DD	00113	PUSHL	KNOWN-		
			53	DD	00115	PUSHL	NEW		
	68		03	FB	00117	CALLS	#3, UTIL_SLIDE		
	69		50	D0	0011A	MOVL	R0, KNOWN_LIST		
	52		53	D0	0011D	MOVL	NEW, KNOWN		1704
			CD	11	00120	BRB	15\$		1691
	54		6A	D0	00122	MOVL	CTL\$GL_PCB, R4		1709

SHOWLOG
V04-000

D 4
16-Sep-1984 01:28:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:09:37 [CLIUTL.SRC]SHOWLOG.B32;1

Page 59
(11)

S
V

12	00000000G	00	16	00125	JSB	LNMSUNLOCK
50		00	DA	0012B	MTPR	#0, #18
		01	DO	0012E	MOVL	#1, R0
		04	00131	RET		

:
: 1710
: 1712
: 1713

; Routine Size: 306 bytes. Routine Base: \$CODES + 09FE

```
1627 1714 1 routine translate(
1628 1715 1     name_len,
1629 1716 1     name_adr,
1630 1717 1     p_known      ! Address of pointer to a known_block
1631 1718 1 ) =
1632 1719 2 begin
1633 1720 2
1634 1721 2     ! This routine translates a logical name in a particular table, and returns
1635 1722 2     ! a pointer to a log_block (or 0).
1636 1723 2
1637 1724 2 local
1638 1725 2     proc_nt:      nt_block,
1639 1726 2     syst_nt:      nt_block,
1640 1727 2     known:        ref known_block,
1641 1728 2     log:           ref log_block,
1642 1729 2     status;
1643 1730 2 local
1644 1731 2     nt:           ref nt_block;                ! Usually R3
1645 1732 2 label
1646 1733 2     main;
1647 1734 2
1648 1735 2 runs_in_kernel (translate);
1649 1736 2 recheck_tables?);
1650 1737 2
1651 1738 2 main:
1652 1739 2 begin
1653 1740 2
1654 1741 2     ! Initialize the values we will return to zero
1655 1742 2
1656 1743 2     known = 0;
1657 1744 2     log = 0;
1658 1745 2
1659 1746 2     ! Initialize the nt_block for the logical name
1660 1747 2
1661 1748 2     decr i from k_process_directory to k_system_directory do
1662 1749 2     begin
1663 1750 2     local
1664 1751 2     lnmb:          ref $bblock;
1665 1752 2
1666 1753 2     if .i eql k_system_directory
1667 1754 2     then nt = syst_nt[base_]
1668 1755 2     else nt = proc_nt[base_];
1669 1756 2
1670 1757 2     ! ??? Note that we set the access mode to .acmode.
1671 1758 2     ! This should be okay, since lnm$presearch and lnm$contsearch only
1672 1759 2     ! check the access mode of the logical name, not the logical name
1673 1760 2     ! table.
1674 1761 2
1675 1762 2     nt[nt_b_acmode] = .acmode;      ! Acceptable integrity mode
1676 1763 2     nt[nt_b_flags] = 0;             ! Initialize flags
1677 1764 2     nt[nt_v_case] = true;           ! Case-insensitive
1678 1765 2     nt[nt_w_hash] = 0;             ! Hash function or 0
1679 1766 2     nt[nt_l_namlen] = .name_len;   ! Length of name
1680 1767 2     nt[nt_l_namadr] = .name_adr;   ! Address of name
1681 1768 2     nt[nt_l_tabid] = 0;            ! Table header address or 0
1682 1769 2     nt[nt_l_thread] = 0;           ! Lnmb pointer
1683 1770 2
```

```
1684 1771 4      ! Do a presearch on the logical name
1685 1772 4
1686 1773 4      status = lnm$presearch(..lnm$al_hashtbl[.i], nt[base_]; lnmb);
1687 1774 4      if not .status then nt[nt_l_thread] = 0;
1688 1775 4      end;
1689 1776 4
1690 1777 4      if .proc_nt[nt_l_thread] eql 0 and .syst_nt[nt_l_thread] eql 0
1691 1778 4      then
1692 1779 4          leave main;
1693 1780 4
1694 1781 4      ! Loop through, trying to find a known table that holds this logical name
1695 1782 4
1696 1783 4      known = .p_known - %fieldexpand(known_l_next,0);
1697 1784 4      while (known = .known[known_l_next]) neq 0 do
1698 1785 4          begin
1699 1786 4              ! We simply put the address of the lnmth block into the nt_l_tabid
1700 1787 4              ! field and call lnm$contsearch. This is okay, because recheck_tables
1701 1788 4              ! has verified that all the tables are still good (or has cleared the
1702 1789 4              ! known_l_lnmth field for those that aren't).
1703 1790 4
1704 1791 4              label
1705 1792 4              lookup:
1706 1793 4              lookup: begin
1707 1794 5                  ! See if the name is in that table.
1708 1795 5
1709 1796 5                  local
1710 1797 5                  table:      ref table_block,
1711 1798 5                  lnmth:      ref $bblock,
1712 1799 5                  lnmb:      ref $bblock,
1713 1800 5                  save_thread;
1714 1801 5                  table = .known[known_l_table];
1715 1802 5                  lnmth = .known[known_l_lnmth];
1716 1803 5                  if lnmth[base_] eql 0 then leave lookup;
1717 1804 5                  if .lnmth[lnmth$shareable]
1718 1805 5                  then nt = syst_nt[base_]
1719 1806 5                  else nt = proc_nt[base_];
1720 1807 5                  nt[nt_l_tabid] = lnmth[base_];
1721 1808 5                  if .nt[nt_l_thread] eql 0 then leave lookup;
1722 1809 5                  ! Now go look for it
1723 1810 5                  save_thread = .nt[nt_l_thread];
1724 1811 5                  status = lnm$contsearch(nt[base_]; lnmb);
1725 1812 5                  nt[nt_l_thread] = .save_thread;
1726 1813 5                  if not .status then leave lookup;
1727 1814 5                  log = new_lnmb(lnmb[base_], table[base_]);
1728 1815 5                  if log[base_] eql 0 then leave lookup;
1729 1816 5                  leave main;
1730 1817 5                  ! Check status
1731 1818 5                  ! Save the lnmb block
1732 1819 5                  ! Verify it's acceptable
1733 1820 5                  ! We found one!
1734 1821 4                  end;      ! of lookup
1735 1822 4
1736 1823 4      ! Now go try another table
1737 1824 4
1738 1825 4      0;
1739 1826 4      end;
1740 1827 4
```

```
: 1741      1828 2      end;          ! of main
: 1742      1829 ~
: 1743      1830 ~      lnm$unlock(.ctl$gl_pcb);
: 1744      1831 ~      set_ipl(0);
: 1745      1832 ~
: 1746      1833 ~      ! Return the address of the log_block,
: 1747      1834 ~      ! and the known_block of the table.
: 1748      1835 ~
: 1749      1836 ~      .p known = known[base_];
: 1750      1837 ~      return log[base_];
: 1751      1838 1      end;
```

```
00          50          01FC 00000 TRANSLATE:
          58 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8
          SE          28 C2 00009 MOVAB CTL$GL_PCB, R8
          50 DC 0000C SUBL2 #40, SP
          02 18 ED 0000E MOVPSL PSL
          0000' CF 07 12 00013 CMPZV #24, #2, PSL, #0
          50 D0 00015 BNEQ 1$
          11 11 0001A MOVL FP, SAVED_FP
          5C DD 0001C 1$: BRB 2$
          DF AF 9F 0001E PUSHL AP
          00000000G 00 02 FB 00021 PUSHAB TRANSLATE
          0000' CF D4 00028 CALLS #2, SYSSCMKRNL
          04 0002C CLRL SAVED_FP
          54 00000000G 68 D0 0002D 2$: MOVL CTL$GL_PCB, R4
          F726 CF 00 16 00030 JSB LNMSLOCKR
          57 D4 0003B CALLS #0, RECHECK_TABLES
          54 01 7D 0003D CLRL LOG
          54 D5 00040 3$: MOVQ #1, I
          05 12 00042 TSTL I
          53 6E 9E 00044 BNEQ 4$
          04 11 00047 MOVAB SYST_NT, NT
          53 14 AE 9E 00049 4$: BRB 5$
          63 0000' CF 9B 0004D 5$: MOVAB PROC_NT, NT
          01 A3 01 88 00052 MOVZBW ACMODE, (NT)
          04 A3 02 A3 B4 00056 BISB2 #1, 1(NT)
          04 04 AC 7D 00059 CLRW 2(NT)
          0C A3 7C 0005E MOVQ NAME_LEN, 4(NT)
          50 00000000G 00 44 D0 00061 CLRQ 12(NT)
          51 00000000G 60 D0 00069 MOVL LNMSAL_HASHTBL[1], R0
          52 50 D0 00072 MOVL (R0), R1
          03 52 E8 00075 JSB LNMSPRESEARCH
          10 A3 D4 00078 MOVL R0, STATUS
          C2 54 F4 0007B 6$: BLBS STATUS, 6$
          24 AE D5 0007E CLRL 16(NT)
          10 05 12 00081 SOBGEQ I, 3$
          4A 13 00086 TSTL PROC_NT+16
          55 0C AC D0 00088 7$: BNEQ 7$
          BEQL SYST_NT+16
          MOVL P_KNOWN, KNOWN
```

1714

1735

1736

1744

1767

1753

1754

1755

1762

1764

1765

1766

1768

1773

1774

1748

1777

1783

55		65	D0 0008C	8\$:	MOVL	(KNOWN), KNOWN	:	1784
		41	13 0008F		BEQL	11\$	:	
56	08	A5	D0 00091		MOVL	8(KNOWN), TABLE	:	1803
50	04	A5	D0 00095		MOVL	4(KNOWN), LNMTN	:	1804
		F1	13 00099		BEQL	8\$	:	1805
05		60	E9 0009B		BLBC	(LNMTN), 9\$	:	1806
53		6E	9E 0009E		MOVAB	SYST_NT, NT	:	1807
		04	11 000A1		BRB	10\$	:	
53	14	AE	9E 000A3	9\$:	MOVAB	PROC_NT, NT	:	1808
OC	A3	50	D0 000A7	10\$:	MOVL	LNMTN, 12(NT)	:	1809
		10	A3 D5 000AB		TSTL	16(NT)	:	1810
		DC	13 000AE		BEQL	8\$	:	
54	10	A3	D0 000B0		MOVL	16(NT), SAVE_THREAD	:	1814
	00000000G	00	16 000B4		JSB	LNMSCONTSEARCH	:	1815
52		50	D0 000BA		MOVL	R0, STATUS	:	
10	A3	54	D0 000BD		MOVL	SAVE_THREAD, 16(NT)	:	1816
	C8	52	E9 000C1		BLBC	STATUS, 8\$	:	1817
	0042	8F	BB 000C4		PUSHR	#M<R1,R6>	:	1818
F827	CF	02	FB 000C8		CALLS	#2, NEW_LNMB	:	
	57	50	D0 000CD		MOVL	R0, LOG	:	
		BA	13 000D0		BEQL	8\$	:	1819
54		68	D0 000D2	11\$:	MOVL	CTL\$GL_PCB, R4	:	1830
	00000000G	00	16 000D5		JSB	LNMSUNLOCK	:	
12		00	DA 000DB		MTPR	#0, #18	:	1831
OC	BC	55	D0 000DE		MOVL	KNOWN, @P_KNOWN	:	1836
	50	57	D0 000E2		MOVL	LOG, R0	:	1837
		04	000E5		RET		:	1838

; Routine Size: 230 bytes, Routine Base: \$CODE\$ + 0B30

```
1753 1839 1 : Routine to recursively translate a logical name.
1754 1840 1
1755 1841 1 : Note that while we are in this routine, we use the virtual memory as a stack
1756 1842 1 : (via util_allocate and util_reallocate).
1757 1843 1
1758 1844 1 routine recursively_translate(
1759 1845 1     name_len,
1760 1846 1     name_adr,
1761 1847 1     depth
1762 1848 1 ): novalue =
1763 1849 2 begin
1764 1850 2 local
1765 1851 2     known: ref known_block,
1766 1852 2     log: ref log_block,
1767 1853 2     lnm: ref $bblock;
1768 1854 2 own
1769 1855 2     transx: vector[lnm$cm_maxdepth];
1770 1856 2
1771 1857 2
1772 1858 2 if .depth gequ lnm$cm_maxdepth then return;
1773 1859 2
1774 1860 2 : Set up for the first known table
1775 1861 2
1776 1862 2 known = known_list - %fieldexpand(known_l_next,0);
1777 1863 2 while (known = .known[known_l_next]) neq 0 do
1778 1864 2     begin
1779 1865 2
1780 1866 2         : Look for the next translation, in a table starting with known.
1781 1867 2         : If a found, known is set to the known_block in which it was found.
1782 1868 2
1783 1869 2         log = translate(.name_len, .name_adr, known);
1784 1870 2         if log[base_] eql 0 then return;
1785 1871 2
1786 1872 2         : Avoid redisplaying recursive translations (in the current path)
1787 1873 2
1788 1874 2         transx[.depth] = log[base_];
1789 1875 2         decr i from .depth-1 to 0 do
1790 1876 2             if .transx[i] eql log[base_] then return;
1791 1877 2
1792 1878 2         : Display this translation
1793 1879 2
1794 1880 2         dump_log(log[base_], .depth);
1795 1881 2
1796 1882 2         : Unlink the log_block from the table.
1797 1883 2         : This isn't really needed. It's just good practice, since the pointer
1798 1884 2         : to the log_block is invalidated by the call to util_reallocate.
1799 1885 2
1800 1886 2     begin
1801 1887 2         local table: ref table_block;
1802 1888 2         table = .log[log_l_table];
1803 1889 2         table[table_l_log] = 0;
1804 1890 2     end;
1805 1891 2
1806 1892 2 : Now loop through, doing recursive translations
1807 1893 2
1808 1894 2 lnm = .log[log_t_name] + 1+log[log_t_name];
1809 1895 2 while not .lnm[lnm$cm_xend] do
```

```

: 1810      1896  4      begin
: 1811      1897  4      if .lnmx[lnmx$b_index] geq 0 and not .lnmx[lnmx$u_terminal]
: 1812      1898  4      then
: 1813      1899  5          begin
: 1814      1900  5              local len,adr;
: 1815      1901  5              len = .lnmx[lnmx$u_xlation];
: 1816      1902  5              adr = i+lnmx[lnmx$u_xlation];
: 1817      1903  5              if .len gequ 4 then if .(adr)<0,16,0> eql %x'001b'
: 1818      1904  5                  then
: 1819      1905  5                      (len = .len - 4; adr = .adr + 4);
: 1820      1906  5
: 1821      1907  5              ! If we have a trailing colon, remove it
: 1822      1908  5              ! (unless there are actually two trailing colons).
: 1823      1909  5              !
: 1824      1910  5              if .len gequ 2 then if
: 1825      1911  5                  ch$rchar(.adr+.len-2) neq ':' and
: 1826      1912  5                  ch$rchar(.adr+.len-1) eql ':'
: 1827      1913  5              then
: 1828      1914  5                  len = .len - 1;
: 1829      1915  5
: 1830      1916  5              recursively_translate(.len, .adr, .depth+1);
: 1831      1917  4              end;
: 1832      1918  4              lnmx = .lnmx[lnmx$u_xlation] + 1+lnmx[lnmx$u_xlation];
: 1833      1919  4              end;
: 1834      1920  3
: 1835      1921  3      ! Back up our memory allocator.
: 1836      1922  3      !
: 1837      1923  3      util_reallocate(log[base_]);
: 1838      1924  2      end;
: 1839      1925  1  end;
```

.PSECT \$OWNS\$,NOEXE,2

0001B .BLKB 1
0001C TRANSX: .BLKB 40

.PSECT \$CODE\$,NOWRT,2

007C 0000 RECURSIVELY TRANSLATE:						
				.WORD	Save R2,R3,R4,R5,R6	: 1844
5E		04	C2 00002	SUBL2	#4, SP	
55	0C	AC	D0 00005	MOVL	DEPTH, R5	: 1858
0A		55	D1 00009	CMPL	R5, #10	
		01	1F 0000C	BLSSU	1\$	
			04 0000E	RET		
6E	0000'	CF	9E 0000F 1\$:	MOVAB	KNOWN LIST, KNOWN	: 1862
56	FF	A5	9E 00014	MOVAB	-1(R5), R6	: 1875
		9E	DD 00018 2\$:	PUSHL	@KNOWN	: 1863
		22	13 0001A	BEQL	4\$	
		5E	DD 0001C	PUSHL	SP	: 1869
7E	04	AC	7D 0001E	MOVQ	NAME_LEN, -(SP)	
FEF3	CF	03	FB 00022	CALLS	#3, TRANSLATE	
	53	50	D0 00027	MOVL	R0, LOG	

	0000'CF45	12	13	0002A	BEQL	4\$	: 1870
	50	53	D0	0002C	MOVL	LOG, TRANSX[R5]	: 1874
		01	A6	9E 00032	MOVAB	1(R6), I	: 1875
	53	08	11	00036	BRB	5\$	: 1876
		0000'CF40	D1	00038	CMPL	TRANSX[I], LOG	: 1876
	F5	77	13	0003E	BEQL	11\$	: 1880
		50	F4	00040	SOBGEQ	I, 3\$	: 1888
		28	BB	00043	PUSHR	#4M<R3,R5>	: 1889
	FC38	02	FB	00045	CALLS	#2, DUMP LOG	: 1894
	50	04	A3	D0 0004A	MOVL	4(LOG), TABLE	: 1895
		10	A0	D4 0004E	CLRL	16(TABLE)	: 1897
	50	0A	A3	9A 00051	MOVZBL	10(LOG) R0	: 1901
	52	0B	A340	9E 00055	MOVAB	11(LOG)[R0], LNMX	: 1902
4F	62	02	E0	0005A	BBS	#2, (LNMX), 10\$	: 1903
		01	A2	95 0005E	TSTB	1(LNMX)	: 1905
		3F	19	00061	BLSS	9\$	: 1910
3B	62	01	E0	00063	BBS	#1, (LNMX), 9\$	: 1911
	54	04	A2	9A 00067	MOVZBL	4(LNMX), LEN	: 1912
	50	05	A2	9E 0006B	MOVAB	5(R2), ADR	: 1914
	04	54	D1	0006F	CMPL	LEN, #4	: 1916
		0B	1F	00072	BLSSU	7\$	: 1918
	1B	60	B1	00074	CMPL	(ADR), #27	: 1919
		06	12	00077	BNEQ	7\$	: 1920
	54	04	C2	00079	SUBL2	#4, LEN	: 1921
	50	04	C0	0007C	ADDL2	#4, ADR	: 1922
	02	54	D1	0007F	CMPL	LEN, #2	: 1923
		12	1F	00082	BLSSU	8\$	: 1924
51	50	54	C1	00084	ADDL3	LEN, ADR, R1	: 1925
	3A	FE	A1	91 00088	CMPL	-2(R1), #58	: 1926
		08	13	0008C	BEQL	8\$	: 1927
	3A	FF	A1	91 0008E	CMPL	-1(R1), #58	: 1928
		02	12	00092	BNEQ	8\$	: 1929
		54	D7	00094	DECL	LEN	: 1930
		01	A5	9F 00096	PUSHAB	1(R5)	: 1931
		50	DD	00099	PUSHL	ADR	: 1932
		54	DD	0009B	PUSHL	LEN	: 1933
FF5E	CF	03	FB	0009D	CALLS	#3, RECURSIVELY_TRANSLATE	: 1934
	50	04	A2	9A 000A2	MOVZBL	4(LNMX) R0	: 1935
	52	05	A240	9E 000A6	MOVAB	5(LNMX)[R0], LNMX	: 1936
		AD	11	000AB	BRB	6\$	: 1937
		53	DD	000AD	PUSHL	LOG	: 1938
F38A	CF	01	FB	000AF	CALLS	#1, UTIL_REALLOCATE	: 1939
		FF61	31	000B4	BRW	2\$	: 1940
		04	000B7	11\$:	RET		: 1941

; Routine Size: 184 bytes, Routine Base: \$CODE\$ + 0C16

```
1841 1926 1 ! This is the main routine
1842 1927 1
1843 1928 1 global routine show$logical =
1844 1929 2 begin
1845 1930 2 local
1846 1931 2 list: ref cmdtbl_block;
1847 1932 2 local
1848 1933 2 free_pages,
1849 1934 2 iosb: vector[2],
1850 1935 2 jpi_list: vector[4] initial(jpi$freptcnt*16+4,free_pages,0.0);
1851 1936 2
1852 1937 2 macro
1853 1938 2 dsc_lenadr_(x) = .x[dsc$w_length], .x[dsc$a_pointer] %;
1854 1939 2
1855 1940 2 ! Initialize all the flags to off
1856 1941 2
1857 1942 2 ch$fill(0, %allocation(flags), flags[base_]);
1858 1943 2 saved_fp = 0;
1859 1944 2
1860 1945 2 ! Allocate virtual memory
1861 1946 2
1862 1947 2 $check_($getjpiw(efn=0,iosb=iosb,itmlst=jpi_list));
1863 1948 2 vm_size = minu((.sgn$gl_pagedyn + 1023),((.free_pages-32)*512));
1864 1949 2 $check_(lib$get_vm(vm_size, vm_addr));
1865 1950 2
1866 1951 2 !+
1867 1952 2 !- Collect the qualifiers and logical names to be translated
1868 1953 2
1869 1954 2
1870 1955 2 ! Determine which standard logical name tables to search
1871 1956 2 ! Tables specified by the /TABLE qualifier are searched before
1872 1957 2 ! the tables that correspond to these qualifiers.
1873 1958 2 ! If none of these are specified, all four tables are searched.
1874 1959 2
1875 1960 2 flags[flag_v_system] = cli$present(%ascid 'system');
1876 1961 2 flags[flag_v_group] = cli$present(%ascid 'group');
1877 1962 2 flags[flag_v_job] = cli$present(%ascid 'job');
1878 1963 2 flags[flag_v_proc] = cli$present(%ascid 'process');
1879 1964 2
1880 1965 2 ! Indicate that the heirarchical structure of the tables should be shown
1881 1966 2
1882 1967 2 flags[flag_v_normal]= not cli$present(%ascid 'structure');
1883 1968 2
1884 1969 2 ! Indicates descendants of the specified tables should also be searched
1885 1970 2
1886 1971 2 flags[flag_v_descend] = cli$present(%ascid 'descendants');
1887 1972 2
1888 1973 2 ! Indicates that lots of information should be displayed
1889 1974 2
1890 1975 2 flags[flag_v_full] = cli$present(%ascid 'full');
1891 1976 2
1892 1977 2 ! Determine the qualifying integrity mode
1893 1978 2 ! Only logical names equal or inner to the qualifying mode are shown.
1894 1979 2 ! The integrity mode is also used to qualify which tables are searched,
1895 1980 2 ! and which tables are shown by the /STRUCTURE qualifier.
1896 1981 2 ! Also, turn on inner integrity mode flags
1897 1982 2 ! If none of the mode qualifiers were specified, assume user mode or inner.
```

```
1898 1983 1
1899 1984 1
1900 1985 1
1901 1986 1
1902 1987 1
1903 1988 1
1904 1989 1
1905 1990 1
1906 1991 1
1907 1992 1
1908 1993 1
1909 1994 1
1910 1995 1
1911 1996 3
1912 1997 4
1913 1998 4
1914 1999 4
1915 2000 4
1916 2001 4
1917 2002 4
1918 2003 5
1919 2004 5
1920 2005 5
1921 2006 5
1922 2007 5
1923 2008 5
1924 2009 5
1925 2010 6
1926 2011 6
1927 2012 6
1928 2013 5
1929 2014 4
1930 2015 4
1931 2016 3
1932 2017 2
1933 2018 2
1934 2019 2
1935 2020 2
1936 2021 2
1937 2022 2
1938 2023 2
1939 2024 2
1940 2025 2
1941 2026 2
1942 2027 2
1943 2028 2
1944 2029 2
1945 2030 3
1946 2031 3
1947 2032 3
1948 2033 3
1949 2034 3
1950 2035 4
1951 2036 4
1952 2037 4
1953 2038 4
1954 2039 4

!
begin
own
  mode_str: vector[4] psect($plit$) preset(
    [psl$kernel]= %ascid 'KERNEL MODE',
    [psl$exec]= %ascid 'EXECUTIVE MODE',
    [psl$super]= %ascid 'SUPERVISOR MODE',
    [psl$user]= %ascid 'USER MODE';
bind
  dsc_access_mode = %ascid 'access_mode';

acmode = psl$user;
if cli$present(dsc_access_mode)
then
  begin
  local
    desc: %bblock[dsc$d_bln];
    $init_dyndesc(desc);
    cli$get_value(dsc_access_mode, desc[base_]);
    decr mode from psl$user to psl$kernel do
    begin
    bind
      name = .mode_str[.mode]: %bblock[dsc$s_bln];
      if .desc[dsc$w_length] lequ .name[dsc$w_length] then
      if ch$eq( .desc[dsc$w_length], .desc[dsc$a_pointer],
        .desc[dsc$w_length], .name[dsc$a_pointer] )
      then
      begin
      acmode = .mode;
      exitloop;
      end;
    end;
    str$free1_dx(desc[base_]);
  end;
end;

!+
Get the table names specified from the command line (cmdtbl_list)
Get the logical names specified from the command line (cmdnam_list)
!-

cmdtbl_list = get_tables();
cmdnam_list = get_lognam();

! Determine whether recursive translations are desired.
! We will show recursive translations if NONE of the logical names
! contain wildcard characters.

begin
local list: ref cmdtbl_block;
list = cmdnam_list - %fieldexpand(cmdtbl_l_next,0);
flags[flag_v_recurse] = true;
while (list = .list[cmdtbl_l_next]) neq 0 do
begin
bind name = list[cmdtbl_q_name]: %bblock[dsc$d_bln];
macro contains(x,y) = not ch$fail(ch$find_ch(dsc$lenadr(x), y)) %;
if contains(name, '*') or contains(name, '%')
then
```

```
1955 2040 5      begin
1956 2041 5      flags[flag_v_recurse] = false;
1957 2042 5      exitloop;
1958 2043 4      end;
1959 2044 3      end;
1960 2045 2      end;
1961 2046 2
1962 2047 2
1963 2048 2      ! Find the relevant tables
1964 2049 2
1965 2050 2      $check(get_relevant_tables());
1966 2051 2      if known_list[base_] eq 0
1967 2052 2      then
1968 2053 3      begin
1969 2054 3          ! Complain if we found no logical name tables
1970 2055 3
1971 2056 3          signal(ss$_ivlogtab and not sts$m_severity or sts$k_warning);
1972 2057 3      end;
1973 2058 2
1974 2059 2      ! Check for recursive translations
1975 2060 2
1976 2061 2      if .flags[flag_v_recurse]
1977 2062 2      then
1978 2063 3      begin
1979 2064 3      local
1980 2065 3      list:      ref cmdtbl_block;
1981 2066 3      list = cmdnam_list - %fieldexpand(cmdtbl_l_next,0);
1982 2067 3      while (list = .list[cmdtbl_l_next]) neq 0 do
1983 2068 3      begin
1984 2069 4      bind name = list[cmdtbl_q_name]: $bblock[dsc$c_d_bln];
1985 2070 4      if .flags[flag_v_found] then write_blank_line();
1986 2071 4      flags[flag_v_found] = false;
1987 2072 4      recursively_translate(dsc_lenadr(name), 0);
1988 2073 4      if not .flags[flag_v_found]
1989 2074 4      then
1990 2075 4          signal(show$_notran,1,list[cmdtbl_q_name]);
1991 2076 4      end;
1992 2077 3      end;
1993 2078 2      else
1994 2079 3      begin
1995 2080 3      if .flags[flag_v_normal]
1996 2081 3      then
1997 2082 3          for_each_lnmb( match_lnmb, 0 );
1998 2083 3
1999 2084 3      ! Now display the tables, and the logical names in the tables
2000 2085 3
2001 2086 3      list = cmdtbl_list - %fieldexpand(cmdtbl_l_next,0);
2002 2087 3      while (list = .list[cmdtbl_l_next]) neq 0 do
2003 2088 3      begin
2004 2089 4      visit_tree(.list[cmdtbl_l_table],
2005 2090 4          %fieldexpand(table_l_child,0),
2006 2091 4          %fieldexpand(table_l_next,0),
2007 2092 4          dump_table,
2008 2093 4          0);
2009 2094 4      end;
2010 2095 3      end;
2011 2096 3
```

```
2012 2097
2013 2098
2014 2099
2015 2100
2016 2101
2017 2102
2018 2103
2019 2104
2020 2105
2021 2106
2022 2107
2023 2108
2024 2109
2025 2110
2026 2111
2027 2112
2028 2113
2029 2114
2030 2115
2031 2116
2032 2117
2033 2118
2034 2119
2035 2120
2036 2121
2037 2122
2038 2123
2039 2124
2040 2125
2041 2126
2042 2127
2043 2128
2044 2129
2045 2130

: If none of the logical names translated, complain about it.
: if .flags[flag_v_normal] and not .flags[flag_v_found]
: then
:   signal(show$notran,1,cmdnam_list[cmdtbl_q_name]);
: end;

: Free the dynamic strings that have been holding table and logical names.
decr i from 1 to 0 do
  begin
    if .i
    then list = cmdtbl_list - %fieldexpand_(cmdtbl_l_next,0)
    else list = cmdnam_list - %fieldexpand_(cmdtbl_l_next,0);
    while (list = .list[cmdtbl_l_next]) neq 0 do
      begin
        bind name = list[cmdtbl_q_name]: $bblock[dsc$d_bln];
        if .name[dsc$b_class] eql dsc$k_class_d
        then
          str$free1_dx(name[base_]);
        end;
      end;
    end;

: Deallocate virtual memory
:
vm_size = minu((.sgn$gl_pagedyn + 1023),((.free_pages-32)*512));
lib$free_vm(vm_size, vm_addr);

return ss$normal;
end;
L1:1948
: Referenced LOCAL symbol FREE_PAGES is probably not initialized
```

```
.PSECT $SPLITS,NOWRT,NOEXE,2

00000000 00000000 04150004 001F1
00 00 6D 65 74 73 79 73 001F4 P.ACH: .BLKB 3
010E0006 0020C P.ACJ: .LONG 68485124
00000000 00210 P.ACI: .LONG 0, 0, 0
010E0005 00214 P.ACL: .ASCII \system\<0><0>
00000000 0021C P.ACK: .LONG 17694726
00 62 6F 6A 00220 .ADDRESS P.ACJ
010E0003 00224 P.ACN: .ASCII \group\<0><0><0>
00000000 00228 P.ACM: .LONG 17694725
00 73 73 65 63 6F 72 70 0022C .ADDRESS P.ACL
010E0007 00230 P.ACP: .ASCII \job\<0>
00000000 00238 P.ACO: .LONG 17694723
00 00 00 65 72 75 74 63 75 72 74 73 0023C .ADDRESS P.ACN
00240 P.ACR: .ASCII \process\<0>
0023C .LONG 17694727
00240 P.ACR: .ADDRESS P.ACP
0023C .ASCII \structure\<0><0><0>
```

```
00 73 74 6E 61 64 6E 65 63 73 65 64 0024C P.ACQ: .LONG 17694729
00000000' 00250 .ADDRESS P.ACR
00 45 44 4F 4D 5F 4C 45 4E 52 45 4B 00254 P.ACT: .ASCII \descendants\<0>
010E000B 00260 P.ACS: .LONG 17694731
00000000' 00264 .ADDRESS P.ACT
6C 6C 75 66 00268 P.ACV: .ASCII \full\
010E0004 0026C P.ACU: .LONG 17694724
00000000' 00270 .ADDRESS P.ACV
00 45 44 4F 4D 5F 4C 45 4E 52 45 4B 00274 P.ACX: .ASCII \KERNEL_MODE\<0>
010E000B 00280 P.ACW: .LONG 17694731
00000000' 00284 .ADDRESS P.ACX
00 45 44 4F 4D 5F 45 56 49 54 55 43 45 58 45 00288 P.ACZ: .ASCII \EXECUTIVE_MODE\<0><0>
00 00297
010E000E 00298 P.ACY: .LONG 17694734
00000000' 0029C .ADDRESS P.ACZ
45 44 4F 4D 5F 52 4F 53 49 56 52 45 50 55 53 002A0 P.ADB: .ASCII \SUPERVISOR_MODE\<0>
00 002AF
010E000F 002B0 P.ADA: .LONG 17694735
00000000' 002B4 .ADDRESS P.ADB
00 00 00 45 44 4F 4D 5F 52 45 53 55 002B8 P.ADD: .ASCII \USER_MODE\<0><0><0>
010E0009 002C4 P.ADC: .LONG 17694729
00000000' 002C8 .ADDRESS P.ADD
00000000' 002CC MODE_STR:
00 65 64 6F 6D 5F 73 73 65 63 63 61 002DC P.ADF: .ADDRESS P.ACW, P.ACY, P.ADA, P.ADC
010E000B 002E8 P.ADE: .ASCII \access_mode\<0>
00000000' 002EC .LONG 17694731
. ADDRESS P.ADF
```

DSC_ACCESS_MODE= P.ADE
.EXTRN SYSSGETJPIW

.PSECT \$CODE\$,NOWRT,2

OFFC 00000

```
5B 00000000G 00 9E 00002
5A 00000000G 00 9E 00009
59 00000000G 00 9E 00010
58 0000' CF 9E 00017
57 00000000G 00 9E 0001C
56 0000' CF 9E 00023
5E 24 C2 00028
OC AE FFOC 08 10 28 0002B
AE 6E 9E 00032
66 B4 00036
FO A6 D4 00038
7E 7C 0003B
24 AE 9F 0003D
18 AE 9F 00040
7E 7C 00043
7E D4 00045
00000000G 00 07 FB 00047
05 50 E8 0004E
50 DD 00051
69 01 FB 00053
50 6B 00003FF 8F C1 00056 1$:
55 6E 20 C3 0005E
```

```
.ENTRY SHOWSLOGICAL, Save R2,R3,R4,R5,R6,R7,R8,R9,-; 1928
R10,R11
MOVAB SGN$GL PAGEDYN, R11
MOVAB L$SIGNAL, R10
MOVAB L$STOP, R9
MOVAB DSC_ACCESS_MODE, R8
MOVAB CL$PRESENT, R7
MOVAB FLAGS, R6
SUBL2 #36, SP
MOVAB #16, P.ACH, JPI_LIST 1935
MOVAB FREE_PAGES, JPI_LIST+4 1929
CLRW FLAGS 1942
CLRL SAVED_FP 1943
CLRQ -(SP) 1947
PUSHAB IOSB
PUSHAB JPI_LIST
CLRQ -(SP)
CLRL -(SP)
CALLS #7, SYSSGETJPIW
BLBS S, 1$
PUSHL S
CALLS #1, LIB$STOP
ADDL3 #1023, SGN$GL PAGEDYN, R0 1948
SUBL3 #32, FREE_PAGES, R5
```

55	55	09 78 00062	ASHL	#9, R5, R5	
	55	50 D1 00066	CMPL	R0, R5	
	50	03 1B 00069	BLEQU	2\$	
	E8 A6	55 D0 0006B	MOVL	R5, R0	
		50 D0 0006E	MOVL	R0, VM SIZE	
		A6 9F 00072	PUSHAB	VM ADDR	1949
		A6 9F 00075	PUSHAB	VM SIZE	
00000000G	00	02 FB 00078	CALLS	#2, LIB\$GET_VM	
	05	50 EB 0007F	BLBS	S, 3\$	
	69	50 DD 00082	PUSHL	S	
		01 FB 00084	CALLS	#1, LIB\$STOP	
	FF24	C8 9F 00087	PUSHAB	P.ACI	1960
	67	01 FB 0008B	CALLS	#1, CLISP\$PRESENT	
66	01	50 FO 0008E	INSV	R0, #0, #1, FLAGS	
		FF34	PUSHAB	P.ACK	1961
	67	01 FB 00097	CALLS	#1, CLISP\$PRESENT	
66	01	50 FO 0009A	INSV	R0, #1, #1, FLAGS	
		FF40	PUSHAB	P.ACM	1962
	67	01 FB 000A3	CALLS	#1, CLISP\$PRESENT	
66	01	50 FO 000A6	INSV	R0, #2, #1, FLAGS	
		FF50	PUSHAB	P.ACO	1963
	67	01 FB 000AF	CALLS	#1, CLISP\$PRESENT	
66	01	50 FO 000B2	INSV	R0, #3, #1, FLAGS	
		FF64	PUSHAB	P.ACO	1967
	67	01 FB 000BB	CALLS	#1, CLISP\$PRESENT	
	51	50 D2 000BE	MCOML	R0, R1	
66	01	51 FO 000C1	INSV	R1, #4, #1, FLAGS	
		FF78	PUSHAB	P.ACS	1971
	67	01 FB 000CA	CALLS	#1, CLISP\$PRESENT	
66	01	50 FO 000CD	INSV	R0, #5, #1, FLAGS	
		84	PUSHAB	P.ACU	1975
	67	01 FB 000D5	CALLS	#1, CLISP\$PRESENT	
66	01	50 FO 000D8	INSV	R0, #6, #1, FLAGS	
	02	03 90 000DD	MOVB	#3, ACMODE	1994
	A6	58 DD 000E1	PUSHL	R8	1995
	67	01 FB 000E3	CALLS	#1, CLISP\$PRESENT	
	41	50 E9 000E6	BLBC	R0, 7\$	
	04 AE 020E0000	8F D0 000E9	MOVL	#344/1936, DESC	2000
		08 AE D4 000F1	CLRL	DESC+4	
		04 AE 9F 000F4	PUSHAB	DESC	2001
		58 DD 000F7	PUSHL	R8	
00000000G	00	02 FB 000F9	CALLS	#2, CLISP\$GET_VALUE	
	54	03 D0 00100	MOVL	#3, MODE	2006
	50	E4 A844 D0 00103	MOVL	MODE_STR[MODE], R0	2005
	60	04 AE B1 00108	CMPL	DESC, (R0)	2006
		0F 1A 0010C	BGTRU	5\$	
04 B0	08 BE	04 AE 29 0010E	CMPC3	DESC, @DESC+4, @4(R0)	2007
		06 12 00115	BNEQ	5\$	
	02 A6	54 90 00117	MOVB	MODE, ACMODE	2011
		03 11 0011B	BRB	6\$	2010
	E3	54 F4 0011D	SOBGEQ	MODE, 4\$	2002
		04 AE 9F 00120	PUSHAB	DESC	2015
00000000G	00	01 FB 00123	CALLS	#1, STR\$FREE1 DX	
F352	CF	00 FB 0012A	CALLS	#0, GET TABLES	2023
F8	A6	50 D0 0012F	MOVL	R0, CMDTBL_LIST	
F3ED	CF	00 FB 00133	CALLS	#0, GET LOGNAM	2024
FC	A6	50 D0 00138	MOVL	R0, CMDNAM_LIST	

		53	FC	A6	9E	0013C	MOVAB	CMDNAM_LIST, LIST	:	2032
		66	80	8F	88	00140	BICB2	#128, FLAGS	:	2033
		53		63	D0	00144	8\$: MOVL	(LIST), LIST	:	2034
				22	13	00147	BEQL	12\$	:	
		52	08	A3	9E	00149	MOVAB	8(LIST), R2	:	2036
04	B2	62		2A	3A	0014D	LOCC	#42, (R2), @4(R2)	:	2038
				02	12	00152	BNEQ	9\$	:	
				51	D4	00154	CLRL	R1	:	
				51	D5	00156	9\$: TSTL	R1	:	
				0D	12	00158	BNEQ	11\$	:	
04	B2	62		25	3A	0015A	LOCC	#37, (R2), @4.R2,	:	
				02	12	0015F	BNEQ	10\$	:	
				51	D4	00161	CLRL	R1	:	
				51	D5	00163	10\$: TSTL	R1	:	
				DD	13	00165	BEQL	8\$	:	
		66	80	8F	8A	00167	11\$: BICB2	#128, FLAGS	:	2041
	FBC0	CF		00	FB	00168	12\$: CALLS	#0, GET_RELEVANT_TABLES	:	2050
		05		50	E8	00170	BLBS	S, 13\$	:	
				50	DD	00173	PUSHL	S	:	
		69		01	FB	00175	CALLS	#1, LIB\$STOP	:	
			F4	A6	D5	00178	13\$: TSTL	KNOWN_LIST	:	2051
				08	12	0017B	BNEQ	14\$	:	
		7E	0158	8F	3C	0017D	MOVZWL	#344, -(SP)	:	2057
		6A		01	FB	00182	CALLS	#1, LIB\$SIGNAL	:	
				66	95	00185	14\$: TSTB	FLAGS	:	2062
				3F	18	00187	BGEQ	17\$	:	
		52	FC	A6	9E	00189	MOVAB	CMDNAM_LIST, LIST	:	2067
		52		62	D0	0018D	15\$: MOVL	(LIST), LIST	:	2068
				79	13	00190	BEQL	21\$	:	
		53	08	A2	9E	00192	MOVAB	8(LIST), R3	:	2070
		09	01	A6	E9	00196	BLBC	FLAGS+1, 16\$	:	2071
			FD18	C8	9F	0019A	PUSHAB	BLANK LINE	:	
	F18F	CF		01	FB	0019E	CALLS	#1, UTIL OUTPUT	:	
	01	A6		01	8A	001A3	16\$: BICB2	#1, FLAGS+1	:	2072
				7E	D4	001A7	CLRL	-(SP)	:	2073
			04	A3	DD	001A9	PUSHL	4(R3)	:	
		7E		63	3C	001AC	MOVZWL	(R3), -(SP)	:	
	FD94	CF		03	FB	001AF	CALLS	#3, RECURSIVELY_TRANSLATE	:	
		D5	01	A6	E8	001B4	BLBS	FLAGS+1, 15\$	:	2074
			08	A2	9F	001B8	PUSHAB	8(LIST)	:	2076
				01	DD	001BB	PUSHL	#1	:	
			00000000G	8F	DD	001BD	PUSHL	#SHOW\$ NOTRAN	:	
		6A		03	FB	001C3	CALLS	#3, LIB\$SIGNAL	:	
				C5	11	001C6	BRB	15\$	:	2068
08		66		04	E1	001C8	17\$: BBC	#4, FLAGS, 18\$	:	2082
				7E	D4	001CC	CLRL	-(SP)	:	2084
			F601	CF	9F	001CE	PUSHAB	MATCH LNMB	:	
	F6E0	CF		02	FB	001D2	CALLS	#2, FOR EACH LNMB	:	
		52	F8	A6	9E	001D7	18\$: MOVAB	CMDTBL_LIST, LIST	:	2088
		52		62	D0	001DB	19\$: MOVL	(LIST), LIST	:	2089
				13	13	001DE	BEQL	20\$	:	
				7F	D4	001E0	CLRL	-(SP)	:	2091
			FA80	C7	9F	001E2	PUSHAB	DUMP TABLE	:	
		7E		04	7D	001E6	MOVQ	#4, =(SP)	:	
			04	A2	DL	001E9	PUSHL	4(LIST)	:	
	F256	CF		05	FB	001EC	CALLS	#5, VISIT_TREE	:	
				E8	11	001F1	BRB	19\$	:	2089

14		66		04	E1	001F3	20\$:	BBC	#4, FLAGS, 21\$		2100
		10	01	A6	E8	001F7		BLBS	FLAGS+1, 21\$		
7E	FC	A6		08	C1	001FB		ADDL3	#8, CMDNAM_LIST, -(SP)		2102
				01	DD	00200		PUSHL	#1		
			00000000G	8F	DD	00202		PUSHL	#SHOWS NOTRAN		
		6A		03	FB	00208		CALLS	#3, LIB\$SIGNAL		
		53		01	D0	0020B	21\$:	MOVL	#1, I		2109
		06		53	E9	0020E	22\$:	BLBC	I, 23\$		2111
		52	F8	A6	9E	00211		MOVAB	CMDTBL_LIST, LIST		2112
				04	11	00215		BRB	24\$		
		52	FC	A6	9E	00217	23\$:	MOVAB	CMDNAM_LIST, LIST		2113
		52		62	D0	0021B	24\$:	MOVL	(LIST), LIST		2114
				15	13	0021E		BEQL	25\$		
		50	08	A2	9E	00220		MOVAB	8(LIST), R0		2116
		02	03	A0	91	00224		CMPB	3(R0), #2		2117
				F1	12	00228		BNEQ	24\$		
	00000000G	00		50	DD	0022A		PUSHL	R0		2119
				01	FB	0022C		CALLS	#1, STR\$FREE1_DX		
		D6		E6	11	00233		BRB	24\$		2114
		6B	000003FF	53	F4	00235	25\$:	SOBGEQ	I, 22\$		2109
50		55		8F	C1	00238		ADDL3	#1023, SGN\$GL_PAGEDYN, R0		2126
				50	D1	00240		CMPL	R0, R5		
				03	1B	00243		BLEQU	26\$		
		50		55	D0	00245		MOVL	R5, R0		
	E8	A6		50	D0	00248	26\$:	MOVL	R0, VM_SIZE		
			EC	A6	9F	0024C		PUSHAB	VM_ADDR		2127
			E8	A6	9F	0024F		PUSHAB	VM_SIZE		
	00000000G	00		02	FB	00252		CALLS	#2, LIB\$FREE_VM		
		50		01	D0	00259		MOVL	#1, R0		2129
				04		0025C		RET			2130

: Routine Size: 605 bytes, Routine Base: \$CODE\$ + 0CCE

: 2046 2131 1
: 2047 2132 1 end
: 2048 2133 0 eludom

.EXTRN LIB\$SIGNAL, LIB\$STOP

PSECT SUMMARY

Name	Bytes	Attributes
\$OWNS	68 NOVEC, WRT, RD	NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$PLITS	752 NOVEC, NOWRT, RD	NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$CODE\$	3883 NOVEC, NOWRT, RD	EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)

Library Statistics

----- Symbols ----- Pages Processing

SHOWLOG
V04-000

6 5
16-Sep-1984 01:28:17
14-Sep-1984 12:09:37

VAX-11 Bliss-32 V4.0-742
[CLIUTL.SRC]SHOWLOG.B32;1

Page 75
(14)

File	Total	Loaded	Percent	Loaded	Time
\$_\$255SDUA28:[SYSLIB]LIB.L32;1	18619	71	0	1000	00:01.9
\$_\$255SDUA28:[SYSLIB]XPORT.L32;1	590	29	4	252	00:01.0

; Information: 1
; Warnings: 0
; Errors: 0

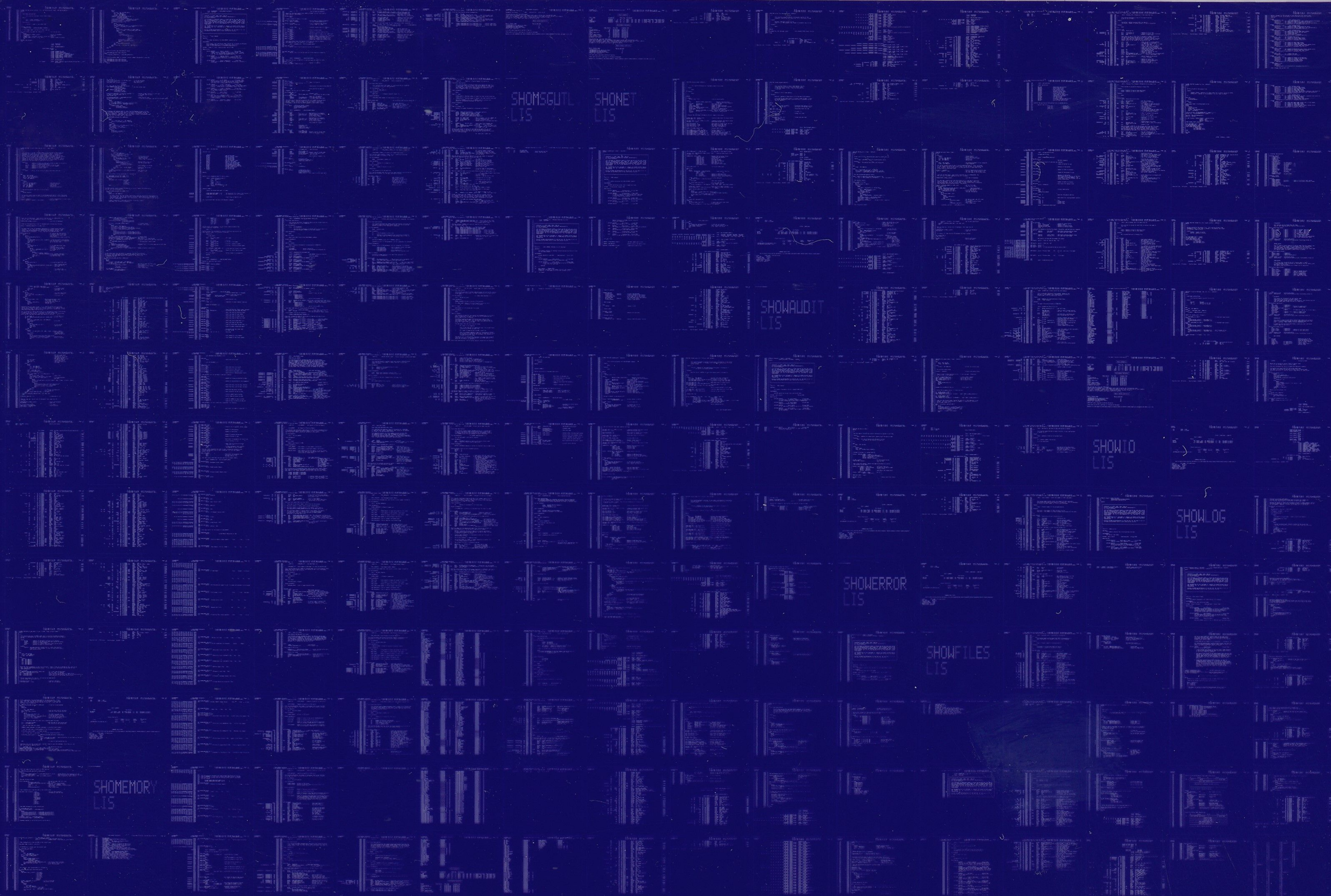
COMMAND QUALIFIERS

; BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:SHOWLOG/OBJ=OBJ\$:SHOWLOG MSRC\$:SHOWLOG/UPDATE=(ENH\$:SHOWLOG)

; Size: 3883 code + 820 data bytes
; Run Time: 01:22.3
; Elapsed Time: 04:01.5
; Lines/CPU Min: 1555
; Lexemes/CPU-Min: 35981
; Memory Used: 272 pages
; Compilation Complete

0056 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY



0057

**DIGITAL
CONFIDE**