

CCCCCCCCCCCC	LLL	IIIIIIII	UUU	UUU	TTTTTTTTTTTTTTTT	LLL
CCCCCCCCCCCC	LLL	IIIIIIII	UUU	UUU	TTTTTTTTTTTTTTTT	LLL
CCCCCCCCCCCC	LLL	IIIIIIII	UUU	UUU	TTTTTTTTTTTTTTTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCC	LLL	III	UUU	UUU	TTT	LLL
CCCCCCCCCCCC	LLLLLLLLLLLLLLLL	IIIIIIII	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	TTTT	LLLLLLLLLLLLLLLL
CCCCCCCCCCCC	LLLLLLLLLLLLLLLL	IIIIIIII	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	TTTT	LLLLLLLLLLLLLLLL
CCCCCCCCCCCC	LLLLLLLLLLLLLLLL	IIIIIIII	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	TTTT	LLLLLLLLLLLLLLLL

```

SSSSSSSS  HH      HH      000000  DDDDDDDD  EEEEEEEEE  VV      VV
SSSSSSSS  HH      HH      000000  DDDDDDDD  EEEEEEEEE  VV      VV
SS        HH      HH      00      00  DD      DD  EE      EE  VV      VV
SS        HH      HH      00      00  DD      DD  EE      EE  VV      VV
SS        HH      HH      00      00  DD      DD  EE      EE  VV      VV
SS        HH      HH      00      00  DD      DD  EE      EE  VV      VV
SSSSSS    HHHHHHHHHH  00      00  DD      DD  EEEEEEEEE  VV      VV
SSSSSS    HHHHHHHHHH  00      00  DD      DD  EEEEEEEEE  VV      VV
          SS      HH      HH      00      00  DD      DD  EE      EE  VV      VV
          SS      HH      HH      00      00  DD      DD  EE      EE  VV      VV
          SS      HH      HH      00      00  DD      DD  EE      EE  VV      VV
          SS      HH      HH      00      00  DD      DD  EE      EE  VV      VV
SSSSSSSS  HH      HH      000000  DDDDDDDD  EEEEEEEEE  VV      VV
SSSSSSSS  HH      HH      000000  DDDDDDDD  EEEEEEEEE  VV      VV

```

```

LL        IIIIII  SSSSSSSS
LL        IIIIII  SSSSSSSS
LL        II     SS
LL        II     SS
LL        II     SS
LL        II     SS
LL        II     SSSSSS
LL        II     SSSSSS
LL        II     SS
LL        II     SS
LL        II     SS
LL        II     SS
LLLLLLLLLL IIIIII  SSSSSSSS
LLLLLLLLLL IIIIII  SSSSSSSS

```

```

1 0001 0 MODULE shodev (IDENT = 'V04-000'
2 0002 0 ADDRESSING_MODE (EXTERNAL = GENERAL)) =
3 0003 0
4 0004 1 BEGIN
5 0005 1
6 0006 1
7 0007 1 *****
8 0008 1 *
9 0009 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
10 0010 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
11 0011 1 * ALL RIGHTS RESERVED.
12 0012 1 *
13 0013 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
14 0014 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
15 0015 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
16 0016 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
17 0017 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
18 0018 1 * TRANSFERRED.
19 0019 1 *
20 0020 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
21 0021 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
22 0022 1 * CORPORATION.
23 0023 1 *
24 0024 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
25 0025 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
26 0026 1 *
27 0027 1 *
28 0028 1 *****
29 0029 1
30 0030 1
31 0031 1 ++
32 0032 1
33 0033 1 FACILITY: SHOW utility
34 0034 1
35 0035 1 ABSTRACT:
36 0036 1 This module contains the main routines for the SHOW commands which
37 0037 1 deal with devices, i.e. SHOW DEVICES, SHOW TERMINAL, SHOW MAGTAPE,
38 0038 1 and SHOW PRINTER.
39 0039 1
40 0040 1 ENVIRONMENT:
41 0041 1 VAX native, user mode.
42 0042 1
43 0043 1 AUTHOR: Gerry Smith CREATION DATE: 28-Jul-1982
44 0044 1
45 0045 1 MODIFIED BY:
46 0046 1
47 0047 1 V03-010 AEW0002 Anne E. Warner 10-Jul-1984
48 0048 1 Add the call to SHOW$MSCP for when the qualifier
49 0049 1 /SERVED is issued. This is the request for the
50 0050 1 display of MSCP served devices which is separate from
51 0051 1 the rest of the SHOW DEVICE code so all this code
52 0052 1 does is call the MSCP code when the qualifier is present
53 0053 1
54 0054 1 V03-009 CWH3009 CW Hobbs 12-Apr-1984
55 0055 1 Add another check for NOSUCHDEV, and add an extra
56 0056 1 blank line for full displays.
57 0057 1

```

58	0058	1	V03-008	CWH3008	CW Hobbs	3-Mar-1984
59	0059	1		Add two routines to sort the device scratch blocks into a list sorted by device name. Change the display loops to follow the list.		
60	0060	1				
61	0061	1				
62	0062	1				
63	0063	1	V03-007	AEW0001	Anne E. Warner	7-Mar-1984
64	0064	1		Fix SHOW DEVICE/WINDOWS so that it automatically sets the /FILES flag and gets to SHOW\$FILES module for processing.		
65	0065	1				
66	0066	1				
67	0067	1	V03-006	CWH3006	CW Hobbs	28-Feb-1984
68	0068	1		Increase virtual buffer so that approximately 1200 devices can be displayed. Change device name parsing logic so that allocation class names (e.g. \$255\$DUA) are accepted. Some other minor cleanups related to dual-path support.		
69	0069	1				
70	0070	1				
71	0071	1				
72	0072	1				
73	0073	1	V03-005	GAS0181	Gerry Smith	19-Sep-1983
74	0074	1		Make it possible for JCP to call the routines necessary to display journals just as SHOW does.		
75	0075	1				
76	0076	1				
77	0077	1	V03-004	GAS0114	Gerry Smith	1-Apr-1983
78	0078	1		Change the display for long device names, so that no special logic is required at this point.		
79	0079	1				
80	0080	1				
81	0081	1	V03-003	GAS0110	Gerry Smith	28-Feb-1983
82	0082	1		Add support for cluster devices.		
83	0083	1				
84	0084	1	V03-002	GAS0107	Gerry Smith	11-Feb-1983
85	0085	1		Add support for journals.		
86	0086	1				
87	0087	1	V03-001	GAS00104	Gerry Smith	17-Jan-1983
88	0088	1		Always initialize the device descriptor and unit number.		
89	0089	1				
90	0090	1				--

```
.. 92 0091 1
.. 93 0092 1
.. 94 0093 1  Include files
.. 95 0094 1
.. 96 0095 1
.. 97 0096 1  LIBRARY 'SYSSLIBRARY:LIB';           ! VAX/VMS system definitions
.. 98 0097 1  REQUIRE 'SRCS:SHOWDEF';           ! SHOW common definitions
.. 99 0196 1  REQUIRE 'SRCS:SHODEVDEF';           ! SHOW DEVICES common definitions
100 0487 1
101 0488 1
102 0489 1
103 0490 1  Macro to set up two associated tables. The first table is a list of
104 0491 1  device types. The second table is a corresponding list of addresses
105 0492 1  of device-specific display routines.
106 0493 1
107 0494 1  If a new device type is to be added, all that is required in this
108 0495 1  macro is to add one line of code, the device type and the corresponding
109 0496 1  display routine name. You must also write the display routine.
110 0497 1
111 0498 1  MACRO
112 0499 1
113 0500 1      device_type [devtype, disprout] = devtype%,
114 0501 1
115 0502 1      display_routine [devtype, disprout] = %NAME(disprout)%,
116 0503 1
117 M 0504 1  make_table (name) =
118 M 0505 1      [ LITERAL device_table_length = (%LENGTH - 1)/2;
119 M 0506 1      EXTERNAL ROUTINE display_routine(%REMAINING);
120 M 0507 1      OWN
121 M 0508 1          device_table : VECTOR[device_table_length, BYTE]
122 M 0509 1          INITIAL (BYTE (device_type(%REMAINING))),
123 M 0510 1
124 M 0511 1          routine_table : VECTOR[device_table_length]
125 M 0512 1          INITIAL (display_routine(%REMAINING));%
.. 126 0513 1
```

```

: 128      0514 1  |
: 129      0515 1  | Set up a table of all device types which have a specific display routine,
: 130      0516 1  | and another table pointing to the address of the display routine each.
: 131      0517 1  |
: 132      P 0518 1  | make_table (dummy,
: 133      P 0519 1  |             dc$_disk,          display_disk,
: 134      P 0520 1  |             dc$_tape,         display_magtape,
: 135      P 0521 1  |             dc$_term,        display_terminal,
: 136      P 0522 1  |             dc$_journal,     display_journal,
: 137      0523 1  |             dc$_lp,          display_printer);

```

```
139 0524 1 |
140 0525 1 | Table of contents
141 0526 1 |
142 0527 1 | FORWARD ROUTINE
143 0528 1 |   show$devices : NOVALUE,
144 0529 1 |   show$printer : NOVALUE,
145 0530 1 |   show$magtape : NOVALUE,
146 0531 1 |   show_device : NOVALUE,
147 0532 1 |   sort_devices : NOVALUE,
148 0533 1 |   insert_device : NOVALUE,
149 0534 1 |   parse_device;
150 0535 1 |
151 0536 1 | EXTERNAL ROUTINE
152 0537 1 |   cli$get_value,
153 0538 1 |   cli$present,
154 0539 1 |   lib$get_vm,
155 0540 1 |   ots$cvt_ti_l,
156 0541 1 |   io_scan,
157 0542 1 |   show$files,
158 0543 1 |   show$mscp,
159 0544 1 |   display_brief : NOVALUE,
160 0545 1 |   display_general : NOVALUE,
161 0546 1 |   show$write_line : NOVALUE;
162 0547 1 |
163 0548 1 | EXTERNAL LITERAL cli$_negated;
164 0549 1 |
165 0550 1 | EXTERNAL
166 0551 1 |   kernel_accvio : VECTOR [4, LONG];
```

```

168 0552 1 GLOBAL ROUTINE show$devices : NOVALUE =
169 0553 2 BEGIN
170 0554 2
171 0555 2 :---
172 0556 2 :
173 0557 2 This is the main routine for SHOW DEVICES. It collects the specific
174 0558 2 qualifiers and then goes to the common SHOW_DEVICE subroutine.
175 0559 2 :
176 0560 2 :---
177 0561 2
178 0562 2 LOCAL
179 0563 2     status,                ! General status return
180 0564 2     flags : $BBLOCK[4] INITIAL(0);    ! Options longword
181 0565 2
182 0566 2 If (flags[devi$v_served] = cli$present(%ASCID 'SERVED'))
183 0567 2 THEN                                ! If SHOW DEVICE/SERVED requested
184 0568 2     BEGIN                            ! execute the routine to display
185 0569 2     show$mscp();                    ! information about MSCP-Served
186 0570 2     RETURN;                          ! devices since it is totally different
187 0571 2     END;                              ! from the rest of the SHOW DEVICE code
188 0572 2
189 0573 2 : Collect the qualifiers.
190 0574 2
191 0575 2 flags[devi$v_allocated] = cli$present(%ASCID 'ALLOCATED');
192 0576 2 flags[devi$v_full] = cli$present(%ASCID 'FULL');
193 0577 2 flags[devi$v_mounted] = cli$present(%ASCID 'MOUNTED');
194 0578 2
195 0579 2 flags[devi$v_files] = cli$present(%ASCID 'FILES');
196 0580 2 IF (flags[devi$v_windows] = cli$present(%ASCID 'WINDOWS'))
197 0581 2 THEN
198 0582 2     flags[devi$v_files] = 1;
199 0583 2
200 0584 2 IF .flags[devi$v_files]
201 0585 2 THEN
202 0586 2     BEGIN
203 0587 2     flags[devi$v_system] = (status = cli$present(%ASCID 'SYSTEM'));
204 0588 2     flags[devi$v_user] = (.status EQL cli$negated);
205 0589 2     IF NOT (.flags[devi$v_system] OR    ! If neither /SYSTEM or
206 0590 2     .flags[devi$v_user])              ! /NOSYSTEM, get both
207 0591 2     THEN flags[devi$v_system] = flags[devi$v_user] = 1;
208 0592 2     END;
209 0593 2
210 0594 2
211 0595 2 show_device(flags);                ! Go actually do the work.
212 0596 2
213 0597 2 RETURN;
214 0598 1 END;

```

```

.TITLE SHODEV
.IDENT \V04-000\

.PSECT $PLITS,NOWRT,NOEXE,2

00 00 44 45 56 52 45 53 0000 P.AAB: .ASCII \SERVED\<0><0>
010E0006 00008 P.AAA: .LONG 17694726
00000000' 0000C .ADDRESS P.AAB

```



```

00 00 00 44 45 54 41 43 4F 4C 4C 41 00010 P.AAD: .ASCII \ALLOCATED\<><0><0>
      010E0009 0001C P.AAC: .LONG 17694729
      00000000 00020 .ADDRESS P.AAD
      4C 4C 55 46 00024 P.AAF: .ASCII \FULL\
      010E0004 00028 P.AAE: .LONG 17694724
      00000000 0002C .ADDRESS P.AAF
      00 44 45 54 4E 55 4F 4D 00030 P.AAH: .ASCII \MOUNTED\<>
      010E0007 00038 P.AAG: .LONG 17694727
      00000000 0003C .ADDRESS P.AAH
      00 00 00 53 45 4C 49 46 00040 P.AAJ: .ASCII \FILES\<><0><0>
      010E0005 00048 P.AAI: .LONG 17694725
      00000000 0004C .ADDRESS P.AAJ
      00 53 57 4F 44 4E 49 57 00050 P.AAL: .ASCII \WINDOWS\<>
      010E0007 00058 P.AAK: .LONG 17694727
      00000000 0005C .ADDRESS P.AAL
      00 00 4D 45 54 53 59 53 00060 P.AAN: .ASCII \SYSTEM\<><0>
      010E0006 00068 P.AAM: .LONG 17694726
      00000000 0006C .ADDRESS P.AAN

```

.PSECT \$OWNS,NOEXE,2

```

      43 A1 42 02 01 00000 DEVICE_TABLE:
      00005 .BYTE 1, 2, 66, -95, 67
00000000G 00000000G 00000000G 00000000G 00000000G 00008 ROUTINE_TABLE:
      00008 .ADDRESS DISPLAY_DISK, DISPLAY_MAGTAPE, -
      DISPLAY_TERMINAL, DISPLAY_JOURNAL, -
      DISPLAY_PRINTER

```

```

.EXTRN DISPLAY_DISK, DISPLAY_MAGTAPE
.EXTRN DISPLAY_TERMINAL
.EXTRN DISPLAY_JOURNAL
.EXTRN DISPLAY_PRINTER
.EXTRN CLISGET_VALUE, CLISPRESNT
.EXTRN LIB$GET_VM, OT$SCVT_TI_L
.EXTRN IO_SCAN, SHOW$FILES
.EXTRN SHOW$MSCP, DISPLAY_BRIEF
.EXTRN DISPLAY_GENERAL
.EXTRN SHOW$WRITE_LINE
.EXTRN CLIS_NEGATED, KERNEL_ACCVIO

```

.PSECT \$CODE\$,NOWRT,2

```

      53 0000 000C 00000 .ENTRY SHOW$DEVICES, Save R2,R3 : 0552
      52 00000000G CF 9E 00002 MOVAB P.AAA, R3 :
      7E D4 0000E MOVAB CLISPRESNT, R2 :
      53 DD 00010 CLRL FLAGS : 0553
      62 01 FB 00012 PUSHL R3 : 0566
      01 AE 01 06 50 FO 00015 CALLS #1, CLISPRESNT
      08 50 E9 0001B INSV R0, #6, #1, FLAGS+1
      00000000G 00 00 FB 0001E BLBC R0, 1$
      04 00025 CALLS #0, SHOW$MSCP : 0569
      14 A3 9F 00026 1$: RET : 0568
      01 AE 01 62 01 FB 00029 PUSHAB P.AAC : 0575
      6E 01 00 50 FO 0002C CALLS #1, CLISPRESNT
      20 A3 9F 00031 INSV R0, #0, #1, FLAGS
      PUSHAB P.AAE : 0576

```

6E	01	62 01		01 FB 00034	CALLS #1, CLISPRESNT	
			30	50 FO 00037	INSV R0, #1, #1, FLAGS	
				A3 9F 0003C	PUSHAB P.AAG	0577
6E	01	62 02		01 FB 0003F	CALLS #1, CLISPRESNT	
			40	50 FO 00042	INSV R0, #2, #1, FLAGS	
				A3 9F 00047	PUSHAB P.AAI	0579
6E	01	62 03		01 FB 0004A	CALLS #1, CLISPRESNT	
			50	50 FO 0004D	INSV R0, #3, #1, FLAGS	
				A3 9F 00052	PUSHAB P.AAK	0580
6E	01	62 06		01 FB 00055	CALLS #1, CLISPRESNT	
				50 FO 00058	INSV R0, #6, #1, FLAGS	
		03		50 F9 0005D	BLBC R0, 2\$	
	28	6E		08 88 00060	BISB2 #8, FLAGS	0582
		6E		03 E1 00063	BBC #3, FLAGS, 4\$	0584
			60	A3 9F 00067	PUSHAB P.AAM	0587
6E	01	62 04		01 FB 0006A	CALLS #1, CLISPRESNT	
				50 FO 0006D	INSV STATUS, #4, #1, FLAGS	
		00000000G	8F	51 D4 00072	CLRL R1	0588
				50 D1 00074	CPL STATUS, #CLIS_NEGATED	
				02 12 0007B	BNEQ 3\$	
6E	01	05		51 D6 0007D	INCL R1	
		07		51 FO 0007F	INSV R1, #5, #1, FLAGS	
		03		04 E0 00084	BBS #4, FLAGS, 4\$	0589
		6E		05 E0 00088	BBS #5, FLAGS, 4\$	0590
		6E		30 88 0008C	BISB2 #48, FLAGS	0591
				5E DD 0008F	PUSHL SP	0595
		0000V	CF	01 FB 00091	CALLS #1, SHOW_DEVICE	
				04 00096	RET	0598

; Routine Size: 151 bytes, Routine Base: \$CODE\$ + 0000

```

: 216      0599 1 GLOBAL ROUTINE show$printer : NOVALUE =
: 217      0600 2 BEGIN
: 218      0601 2
: 219      0602 2 |---
: 220      0603 2 |
: 221      0604 2 | This is the dummy routine that gets dispatched to by the SHOW dispatcher.
: 222      0605 2 | It sets the /FULL and /PRINTER bits in FLAGS and calls SHOW_DEVICE.
: 223      0606 2 |
: 224      0607 2 |---
: 225      0608 2
: 226      0609 2 LOCAL flags : $BBLOCK[4] INITIAL(0);
: 227      0610 2
: 228      0611 2 flags[devi$v_full] = flags[devi$v_printer] = true;
: 229      0612 2 show_device(flags);
: 230      0613 2
: 231      0614 2 RETURN;
: 232      0615 1 END;

```

```

                                0000 00000          .ENTRY SHOWSPRINTER, Save nothing          : 0599
                                7E D4 00002          CLRL  FLAGS                               : 0600
                                6E 0102 8F A8 00004    BISW2 #258, FLAGS+1                          : 0611
                                0000V CF 5E DD 00009    PUSHL SP                                    : 0612
                                01 FB 0000B          CALLS #1, SHOW_DEVICE                          : 0613
                                04 00010          RET                                         : 0615

```

: Routine Size: 17 bytes, Routine Base: \$CODE\$ + 0097

```

: 234      0616 1 GLOBAL ROUTINE show$magtape : NOVALUE =
: 235      0617 2 BEGIN
: 236      0618 2
: 237      0619 2 |---
: 238      0620 2 |
: 239      0621 2 | This is the dummy routine that gets dispatched to by the SHOW dispatcher.
: 240      0622 2 | It sets the /TAPE and /FULL bits in FLAGS and calls SHOW_DEVICE.
: 241      0623 2 |
: 242      0624 2 |---
: 243      0625 2
: 244      0626 2 LOCAL flags : $BBLOCK[4] INITIAL(0);
: 245      0627 2
: 246      0628 2 flags[devi$full] = flags[devi$tape] = true;
: 247      0629 2 show_device(flags);
: 248      0630 2
: 249      0631 2 RETURN;
: 250      0632 1 END;

```

```

                                0000 0000      .ENTRY SHOWSMAGTAPE, Save nothing      : 0616
                                7E D4 0002      CLRL  FLAGS                          : 0617
                                6E 0402 8F AB 0004  BISW2 #1026, FLAGS+1          : 0628
                                0000V CF 5E DD 00009  PUSHL SP                          : 0629
                                01 FB 0000B  CALLS #1, SHOW_DEVICE          :
                                04 00010  RET                                     : 0632

```

: Routine Size: 17 bytes. Routine Base: \$CODE\$ + 00A8

```
252 0633 1 GLOBAL ROUTINE show_device (flags, journal) : NOVALUE =
253 0634 BEGIN
254 0635
255 0636 ---
256 0637
257 0638 This is the common routine that all the other routines feed into. It
258 0639 obtains a device name, if any is specified. The device name is parsed,
259 0640 virtual memory is then expanded, and appropriate routines are called in
260 0641 kernel mode to collect the data. Upon return from the kernel mode routines,
261 0642 the common output routine is called.
262 0643
263 0644 Inputs:
264 0645     flags - address of control flags.
265 0646     journal - optional, used only by JCP to pass the name of the
266 0647             journal.
267 0648
268 0649 ---
269 0650
270 0651 BUILTIN
271 0652     actualcount;
272 0653
273 0654 MAP flags : REF $BBLOCK;
274 0655
275 0656 LOCAL
276 0657     status,                                ! General status return
277 0658     unit : VOLATILE,                        ! Unit number of parsed device
278 0659     node : VECTOR[sb$s_nodename+1,BYTE],    ! Node string
279 0660     device : VECTOR[log$sc_namlength+1,BYTE], ! Device string
280 0661     device_desc : $BBLOCK[dsc$sc_s_bln],    ! Device descriptor
281 0662     data : REF VECTOR,                      ! Address of scratch area
282 0663     arglst : VECTOR[?];                    ! Argument list for $CMKRNL
283 0664
284 0665 !
285 0666 ! Initialize the ASCII strings, the unit number, and the device descriptor
286 0667 !
287 0668 node[0] = device[0] = 0;                    ! Nothing in strings yet
288 0669 unit = -1;                                  ! No unit number yet
289 0670 $init_dyndesc(device_desc);                 ! Set up the device descriptor
290 0671
291 0672 !
292 0673 ! If from JCP, set up the device descriptor.
293 0674 !
294 0675 IF actualcount() EQL 2
295 0676 THEN
296 0677     BEGIN
297 0678     MAP journal : REF $BBLOCK;
298 0679     device_desc[dsc$sw_length] = .journal[dsc$sw_length],
299 0680     device_desc[dsc$a_pointer] = .journal[dsc$a_pointer];
300 0681     END
301 0682
302 0683 !
303 0684 ! Otherwise, just a normal path.
304 0685 !
305 0686 ELSE
306 0687     BEGIN
307 0688
308 0689
```

```

309 0690 3 ! If no device name is specified, then certain defaults may take effect.
310 0691 3 SHOW TERMINAL uses SYSSCOMMAND, and SHOW DEV/FILES uses SYSSDISK.
311 0692 3
312 0693 3     IF NOT cli$get_value(%ASCID 'DEVICE', device_desc)
313 0694 3     THEN
314 0695 3         BEGIN
315 0696 3         IF .flags[devi$V_term] ! If SHOW TERMINAL and none
316 0697 3         THEN ! specified, use SYSSCOMMAND
317 0698 3             BEGIN
318 0699 3             device_desc[dsc$w_length] = %CHARCOUNT ('SYSSCOMMAND');
319 0700 3             device_desc[dsc$a_pointer] = UPLIT BYTE ('SYSSCOMMAND');
320 0701 3             END
321 0702 3         ELSE IF .flags[devi$V_files] ! If SHOW DEV/FILES and no disk
322 0703 3         THEN ! then use SYSSDISK
323 0704 3             BEGIN
324 0705 3             device_desc[dsc$w_length] = %CHARCOUNT ('SYSSDISK');
325 0706 3             device_desc[dsc$a_pointer] = UPLIT BYTE ('SYSSDISK');
326 0707 3             END;
327 0708 3         END;
328 0709 3     END;
329 0710 3
330 0711 3
331 0712 3 ! If SHOW DEVICE/FILES was specified, make a major detour, and simply call
332 0713 3 ! the SHOWFILES module. SHOW FILES is just too different from the way that
333 0714 3 ! the rest of SHOW DEVICES works to try to thread it in.
334 0715 3
335 0716 3 IF .flags[devi$V_files]
336 0717 3 THEN
337 0718 3     BEGIN
338 0719 3     status = show$files(device_desc, .flags);
339 0720 3     IF NOT .status
340 0721 3     THEN SIGNAL(.STATUS);
341 0722 3     RETURN;
342 0723 3     END;
343 0724 3
344 0725 3
345 0726 3 ! If, after all this rigamarole, there is actually a device name to parse,
346 0727 3 ! then go ahead and do it.
347 0728 3
348 0729 3 IF .device_desc[dsc$w_length] NEQ 0
349 0730 3 THEN
350 0731 3     BEGIN
351 0732 3     IF NOT (status = parse_device(device_desc, ! Pass device as input
352 0733 3     node, ! Get node part
353 0734 3     device, ! Get DDB part
354 0735 3     unit, ! Get unit number
355 0736 3     .flags)) possibly set flag bits
356 0737 3     THEN (SIGNAL(.status);RETURN); ! Go away if error.
357 0738 3     END;
358 0739 3
359 0740 3
360 0741 3 ! Grab a large chunk of space, to put the information about the device(s).
361 0742 3
362 0743 3 IF NOT (status = lib$get_vm(%REF(512*512), data))
363 0744 3 THEN (SIGNAL(.status); RETURN);
364 0745 3
365 0746 2 data[0] = 512*512; ! Store the size of the segment

```

```
366 0747 2
367 0748 2
368 0749 2  Now get information on the device(s) requested.
369 0750 2
370 0751 2 arglst[0] = 5;
371 0752 2 arglst[1] = node;
372 0753 2 arglst[2] = device;
373 0754 2 arglst[3] = .unit;
374 0755 2 arglst[4] = .flags;
375 0756 2 arglst[5] = .data;
376 P 0757 2 status = $CMKRN(LROUTIN = io_scan,
377 0758 2 ARGST = arglst);
378 0759 2 IF NOT .status
379 0760 2 THEN
380 0761 2 BEGIN
381 0762 2 IF .status EQL ss$_accvio
382 0763 2 THEN SIGNAL(.status, .kernel_accvio[0], .kernel_accvio[1], .kernel_accvio[2], .kernel_accvio[3], 0)
383 0764 2 ELSE SIGNAL(.status);
384 0765 2 RETURN;
385 0766 2 END;
386 0767 2
387 0768 2
388 0769 2  Sort the devices so that the displays are cleaner
389 0770 2
390 0771 2 sort_devices(data[0]);
391 0772 2
392 0773 2
393 0774 2  Print the information. The method that is used is very dumb, but it works.
394 0775 2  The scratch area is scanned repeatedly, once for each device class. If a
395 0776 2  particular device gets printed, its D_V_DISPLAYED bit is set.
396 0777 2
397 0778 2  Then, all the devices that didn't get printed in the device-specific scan
398 0779 2  get printed in a general format.
399 0780 2
400 0781 2 BEGIN
401 0782 2 LOCAL
402 0783 2 scratch : REF $BBLOCK;
403 0784 2
404 0785 2 flags[devi$v_displayed] = 0; ! Assume that no devices will be found
405 0786 2
406 0787 2 Go thru each device type.
407 0788 2
408 0789 3 INCR index FROM 0 TO device_table_length - 1 DO
409 0790 4 BEGIN
410 0791 4 flags[devi$v_header] = 1; ! Print a header the first time
411 0792 4 scratch = .data[0]; ! Point to head of device list
412 0793 4 WHILE .scratch NEQ 0 DO ! Go thru all the devices
413 0794 5 BEGIN ! a device class at a time
414 0795 5 IF .scratch[d_b_devclass] EQLU .device_table[index]
415 0796 5 THEN
416 0797 6 BEGIN
417 0798 7 IF (.flags[devi$v_full]) ! IF /FULL, do dev-specific output
418 0799 6 THEN (.routine_table[index])(.scratch, .flags)
419 0800 6 ELSE display_brief (.scratch, .flags); ! Otherwise use the general output routine.
420 0801 6 scratch[d_v_displayed] = 1; ! So we don't re-print this device.
421 0802 5 END;
422 0803 5 scratch = .scratch[d_l_ucb]; ! Get to next device.
```

```

423 0804 4      END;
424 0805 4      END;
425 0806 4      END;
426 0807 4      END;
427 0808 4      Now to print the general-device stuff.
428 0809 4      END;
429 0810 4      flags[devi$V_header] = true;          ! Get a heading
430 0811 4      scratch = .data[C];                ! Back to the head of the list
431 0812 4      WHILE .scratch NEQ 0 DO
432 0813 4          BEGIN
433 0814 4              IF NOT .scratch[d_v_displayed]
434 0815 4                  THEN
435 0816 4                      BEGIN
436 0817 4                          IF (.flags[devi$V_full])          ! IF /FULL, do dev-specific output
437 0818 4                              THEN display_general(.scratch, .flags)
438 0819 4                              ELSE display_brief (.scratch, .flags); ! Otherwise use the general output routine.
439 0820 4                              scratch[d_v_displayed] = 1;    ! So we don't re-print this device.
440 0821 4                          END;
441 0822 4                      scratch = .scratch[d_l_ucb];        ! Get to next device.
442 0823 4                      END;
443 0824 4      END;
444 0825 4      END;
445 0826 4      END;
446 0827 4      ! If nobody managed to set the displayed bit, then we saw no devices
447 0828 4      END;
448 0829 2      IF NOT .flags[devi$V_displayed]
449 0830 2      THEN
450 0831 2          SIGNAL(SS$NOSUCHDEV)
451 0832 2      ELSE IF .flags[devi$V_full]
452 0833 2      THEN
453 0834 2          show$write_line(%ASCII ' ', flags);
454 0835 2      END;
455 0836 2      END;
456 0837 2      RETURN;
457 0838 1      END;

```

```

                                .PSECT $SPLITS,NOWRT,NOEXE,2
                                00 00 45 43 49 56 45 44 00070 P.AAP: .ASCII \DEVICE\<0><0>
                                010E0006 00078 P.AAO: .LONG 17694726
                                00000000' 0C07C .ADDRESS P.AAP
44 4E 41 4D 4D 4F 43 24 53 59 53 00080 P.AAQ: .ASCII \SYSS$COMMAND\
                                4B 53 49 44 24 53 59 53 0008B P.AAR: .ASCII \SYSS$DISK\
                                00093 .BLKB 1
                                00094 P.AAT: .BLKB 0
                                010E0000 00094 P.AAS: .LONG 17694720
                                00000000' 00098 .ADDRESS P.AAT

                                .EXTRN SYSS$CMKRNL

                                .PSECT $CODE$,NOWRT,2

                                00FC 00000 .ENTRY SHOW_DEVICE, Save R2,R3,R4,R5,R6,R7
                                57 00000000G 00 9E 00002 MOVAB DISPCAY BRIEF, R7
                                56 00000000G 00 9E 00009 MOVAB LIB$SIGNAL, R6

```


		SE	FF78	CE	9E	00010	MOVAB	-136(SP), SP		
			2C	AE	94	00015	CLRB	DEVICE		0668
			70	AE	94	00018	CLRB	NODE		
	FC	AD		01	CE	0001B	MNEGL	#1, UNIT		0669
	24	AE	020E0000	8F	D0	0001F	MOVL	#34471936, DEVICE_DESC		0670
				AE	D4	00027	CLRL	DEVICE_DESC+4		
		02		6C	91	0002A	CMPB	(AP), #2		0675
				0F	12	0002D	BNEQ	1\$		
		50	08	AC	D0	0002F	MOVL	JOURNAL, R0		0679
	24	AE		60	B0	00033	MOVW	(R0), DEVICE_DESC		
	28	AE		A0	D0	00037	MOVL	4(R0), DEVICE_DESC+4		0680
				34	11	0003C	BRB	3\$		0675
				AE	9F	0003E	1\$: PUSHAB	DEVICE_DESC		0693
			0000'	CF	9F	00041	PUSHAB	P.AAQ		
	00000000G	00		02	FB	00045	CALLS	#2, CLISGET_VALUE		
		23		50	E8	0004C	BLBS	R0, 3\$		
		50	04	AC	D0	0004F	MOVL	FLAGS, R0		0696
OC	01	A0		01	E1	00053	BBC	#1, 1(R0), 2\$		
	24	AE		0B	B0	00058	MOVW	#11, DEVICE_DESC		0699
	28	AE	0000'	CF	9E	0005C	MOVAB	P.AAQ, DEVICE_DESC+4		0700
				0E	11	00062	BRB	3\$		0696
OA		60		03	E1	00064	2\$: BBC	#3, (R0), 3\$		0702
	24	AE		08	B0	00068	MOVW	#8, DEVICE_DESC		0705
	28	AE	0000'	CF	9E	0006C	MOVAB	P.AAR, DEVICE_DESC+4		0706
		54	04	AC	D0	00072	3\$: MOVL	FLAGS, R4		0716
13		64		03	E1	00076	BBC	#3, (R4), 4\$		
				54	DD	0007A	PUSHL	R4		0719
				AE	9F	0007C	PUSHAB	DEVICE_DESC		
	00000000G	00		02	FB	0007F	CALLS	#2, SHOWFILES		
		52		50	D0	00086	MOVL	R0, STATUS		
		37		52	E9	00089	BLBC	STATUS, 6\$		0720
					04	0008C	RET			0721
				AE	B5	0008D	4\$: TSTW	DEVICE_DESC		0729
				19	13	00090	BEQL	5\$		
				54	DD	00092	PUSHL	R4		0736
				FC	AD	00094	PUSHAB	UNIT		0732
				34	AE	00097	PUSHAB	DEVICE		
				7C	AE	0009A	PUSHAB	NODE		
				34	AE	0009D	PUSHAB	DEVICE_DESC		
	0000V	CF		05	FB	000A0	CALLS	#5, PARSE_DEVICE		
		52		50	D0	000A5	MOVL	R0, STATUS		
		6E		52	E9	000A8	BLBC	STATUS, 7\$		
			04	AE	9F	000AB	5\$: PUSHAB	DATA		0743
	04	AE	00040000	8F	D0	000AE	MOVL	#262144, 4(SP)		
			04	AE	9F	000B6	PUSHAB	4(SP)		
	00000000G	00		02	FB	000B9	CALLS	#2, LIB\$GET_VM		
		52		50	D0	000C0	MOVL	R0, STATUS		
		53		52	E9	000C3	6\$: BLBC	STATUS, 7\$		
		55	04	AE	D0	000C6	MOVL	DATA, R5		0746
		65	00040000	8F	D0	000CA	MOVL	#262144, (R5)		
	08	AE		05	D0	000D1	MOVL	#5, ARGLST		0751
OC		AE	70	AE	9E	000D5	MOVAB	NODE, ARGLST+4		0752
10		AE	2C	AE	9E	000DA	MOVAB	DEVICE, ARGLST+8		0753
14		AE	FC	AD	D0	000DF	MOVL	UNIT, ARGLST+12		0754
18		AE		54	7D	000E4	MOVQ	R4, ARGLST+16		0755
				AE	9F	000E8	PUSHAB	ARGLST		0758
			00000000G	00	9F	000EB	PUSHAB	IO_SCAN		

00000000G	00	02	FB	000F1	CALLS	#2, SYS\$CMKRN	
	52	50	DO	000F8	MOVL	R0, STATUS	
	1F	52	EB	000FB	BLBS	STATUS, 8\$	0759
	0C	52	D1	000FE	CMPL	STATUS, #12	0762
		16	12	00101	BNEQ	7\$	
		7E	D4	00103	CLRL	-(SP)	0763
	7E	00000000G	00	7D	00105	MOVQ	KERNEL_ACCVIO+8, -(SP)
	7E	00000000G	00	7D	0010C	MOVQ	KERNEL_ACCVIO, -(SP)
			52	DD	00113	PUSHL	STATUS
	66		06	FB	00115	CALLS	#6, LIB\$SIGNAL
			04	00118	RET		
			52	DD	00119	7\$: PUSHL	STATUS
			76	11	0011B	BRB	20\$
			55	DD	0011D	8\$: PUSHL	R5
0000V	CF	01	FB	0011F	CALLS	#1, SORT_DEVICES	0771
01	A4	20	8A	00124	BICB2	#32, 1(R4)	0785
		53	D4	00128	CLRL	INDEX	0789
	01	A4	08	88	0012A	9\$: BISB2	#8, 1(R4)
	52		65	DO	0012E	MOVL	(R5), SCRATCH
			28	13	00131	10\$: BEQL	14\$
0000'CF43		78	A2	91	00133	CMPB	120(SCRATCH), DEVICE_TABLE[INDEX]
			1A	12	0013A	BNEQ	13\$
OD	64		01	E1	0013C	BBC	#1, (R4), 11\$
	50	0000'CF43	14	DO	00140	MOVL	ROUTINE_TABLE[INDEX], R0
			14	BB	00146	PUSHR	#*M<R2,R4>
	60		02	FB	00148	CALLS	#2, (R0)
			05	11	0014B	BRB	12\$
			14	BB	0014D	11\$: PUSHR	#*M<R2,R4>
	67		02	FB	0014F	CALLS	#2, DISPLAY_BRIEF
	04	A2	01	88	00152	12\$: BISB2	#1, 4(SCRATCH)
	52		62	DO	00156	13\$: MOVL	(SCRATCH), SCRATCH
			D6	11	00159	BRB	10\$
CB	53		04	F3	0015B	14\$: AOBLEQ	#4, INDEX, 9\$
	01	A4	08	88	0015F	BISB2	#8, 1(R4)
	52		65	DO	00163	MOVL	(R5), SCRATCH
			21	13	00166	15\$: BEQL	19\$
	18	04	A2	EB	00168	BLBS	4(SCRATCH), 18\$
OB	64		01	E1	0016C	BBC	#1, (R4), 16\$
			14	BB	00170	PUSHR	#*M<R2,R4>
00000000G	00		02	FB	00172	CALLS	#2, DISPLAY_GENERAL
			05	11	00179	BRB	17\$
			14	BB	0017B	16\$: PUSHR	#*M<R2,R4>
	67		02	FB	0017D	CALLS	#2, DISPLAY_BRIEF
	04	A2	01	88	00180	17\$: BISB2	#1, 4(SCRATCH)
	52		62	DO	00184	18\$: MOVL	(SCRATCH), SCRATCH
			DD	11	00187	BRB	15\$
09	01	A4	05	E0	00189	19\$: BBS	#5, 1(R4), 21\$
	7E	0908	8F	3C	0018E	MOVZWL	#2312, -(SP)
	66		01	FB	00193	20\$: CALLS	#1, LIB\$SIGNAL
			04	00196	RET		
OE	64		01	E1	00197	21\$: BBC	#1, (R4), 22\$
		04	AC	9F	0019B	PUSHAB	FLAGS
		0000'	CF	9F	0019E	PUSHAB	P.AAS
00000000G	00		02	FB	001A2	CALLS	#2, SHOWWRITE_LINE
			04	001A9	22\$: RET		0838

; Routine Size: 426 bytes, Routine Base: \$CODE\$ + 00B9

SHODEV
V04-000

B 7
16-Sep-1984 01:32:33
14-Sep-1984 12:09:25

VAX-11 Bliss-32 V4.0-742
[CLIUTL.SRC]SHODEV.B32:1

Page 17
(8)

S
V

```

: 459 0839 1 ROUTINE sort_devices (data : REF VECTOR [, LONG]) : NOVALUE =
: 460 0840 2 BEGIN
: 461 0841 2
: 462 0842 2 |---
: 463 0843 2 |
: 464 0844 2 | This routine links the scratch areas into a sorted list. The sort key is
: 465 0845 2 | the device name. A special key field is used, since if the device name
: 466 0846 2 | per se were to be used we would see DUA10: before DUA2:..
: 467 0847 2 |
: 468 0848 2 |---
: 469 0849 2
: 470 0850 2 LOCAL
: 471 0851 2     scratch : REF $BLOCK,           ! Address of current entry in scratch area
: 472 0852 2     len;                       ! Length of device name
: 473 0853 2
: 474 0854 2 data[0] = 0;                       ! Use the first longword as the list head
: 475 0855 2 scratch = data[1];                 ! Point to start of scratch area
: 476 0856 2 WHILE .scratch[d_t_device] NEQ 0 DO ! Go thru all the devices (name[0,0,8,0] = 0 marks the end)
: 477 0857 2     BEGIN
: 478 0858 2     BIND
: 479 0859 2         dev = scratch[d_t_device] : VECTOR [, BYTE];
: 480 0860 2         len = .scratch[d_b_devlen]-1; ! Get the total length without the colon
: 481 0861 2         DECR i FROM (.len-1) TO 0 ! Adjust length for 0:n-1 index and scan backwards
: 482 0862 2         DO ! through the string, looking for the last
: 483 0863 2             BEGIN ! non-digit in the string. This trims the unit number.
: 484 0864 2             IF .dev[i] GTR %'9' OR .dev[i] LSS %'0'
: 485 0865 2             THEN EXITLOOP; ! Non-digit, done with this one
: 486 0866 2             len = .len - i; ! Found a digit, shorten the string
: 487 0867 2             END;
: 488 0868 2 CHSMOVE(.len, dev, scratch[d_t_sort_name]); ! Move the node/controller to the sort field
: 489 0869 2 insert_device(.scratch, data[0]); ! Insert it in the list
: 490 0870 2 IF .scratch[d_b_devclass] EQLU dc$ journal ! Skip over the journal device
: 491 0871 2 THEN scratch = .scratch + d_k_length;
: 492 0872 2 scratch = .scratch + d_k_length; ! Get the next device.
: 493 0873 2 END;
: 494 0874 2
: 495 0875 2 RETURN;
: 496 0876 1 END;

```

				00FC 0000 SORT_DEVICES:			
				.WORD	Save R2,R3,R4,R5,R6,R7		: 0839
				CLRL	@DATA		: 0854
57	04	AC	04 BC D4 00002	ADDL3	#4, DATA, SCRATCH		: 0855
			08 A7 95 0000A 1\$:	TSTB	8(SCRATCH)		: 0856
			3F 13 0000D	BEQL	6\$		
		56	06 A7 9A 0000F	MOVZBL	6(SCRATCH), LEN		: 0860
		50	76 9E 00013	MOVAB	-(LEN), I		: 0861
			10 11 00016	BRB	3\$		
		39	08 A740 91 00018 2\$:	CMPB	8(SCRATCH)[I], #57		: 0864
			0C 1A 0001D	BGTRU	4\$		
		30	08 A740 91 0001F	CMPB	8(SCRATCH)[I], #48		
			05 1F 00024	BLSSU	4\$		
			56 D7 00026	DECL	LEN		: 0866

44	A7	08	ED A7	50	F4	00028	3\$:	SOBGEQ	1, 2\$:	0861	
				56	28	0002B	4\$:	MOVCS	LEN, 8(SCRATCH), 68(SCRATCH)	:	0868	
				04	AC	DD	00031	PUSHL	DATA	:	0869	
					57	DD	00034	PUSHL	SCRATCH	:		
		0000V	CF	02	FB	00036		CALLS	#2, INSERT DEVICE	:		
		A1	8F	78	A7	91	0003B	CMPB	120(SCRATCH), #161	:	0870	
					05	12	00040	BNEQ	5\$:		
			57	0107	C7	9E	00042	MOVAB	263(R7), SCRATCH	:	0871	
			57	0107	C7	9E	00047	5\$:	MOVAB	263(R7), SCRATCH	:	0872
					BC	11	0004C	BRB	1\$:	0856	
					04	0004E	6\$:	RET		:	0876	

; Routine Size: 79 bytes, Routine Base: \$CODE\$ + 0263

```

498 0877 1 ROUTINE insert_device (new : REF $BLOCK, head : REF $BLOCK) : NOVALUE =
499 0878 BEGIN
500 0879
501 0880
502 0881
503 0882 This routine inserts the input device into the list of sorted device
504 0883 scratch blocks, using the D_L_UCB field as the link.
505 0884
506 0885 Inputs:
507 0886 new - address of device to be added
508 0887
509 0888
510 0889
511 0890 LOCAL
512 0891     nxt : REF $BLOCK, ! Pointer to next device block
513 0892     prv : REF $BLOCK; ! Pointer to last device block
514 0893
515 0894 $ASSUME ($BYTEOFFSET(d_l_uch), EQL, 0); ! Only works if UCB is the first field
516 0895 prv = head[d_l_uch]; ! Previous starts out as the head
517 0896 nxt = .head[d_l_uch]; ! Next starts as the first one
518 0897
519 0898 WHILE 1 ! Use EXITLOOP as a structured GOTO
520 0899 DO
521 0900 BEGIN
522 0901 IF .nxt EQL 0 ! At the end of the list, insert it here
523 0902 THEN
524 0903 4 BEGIN ! (identical blocks, compiler with combine into one)
525 0904 4     prv[d_l_uch] = .new; ! Point last block at this one
526 0905 4     new[d_l_uch] = .nxt; ! Point this block at the next
527 0906 4     EXITLOOP;
528 0907 4     END;
529 0908 IF CH$GTR(d_s_sort_name, nxt[d_t_sort_name], ! If next is greater than the new, insert it here
530 0909     d_s_sort_name, new[d_t_sort_name])
531 0910 THEN
532 0911 4 BEGIN
533 0912 4     prv[d_l_uch] = .new; ! Point last block at this one
534 0913 4     new[d_l_uch] = .nxt; ! Point this block at the next
535 0914 4     EXITLOOP;
536 0915 4     END;
537 0916     prv = .nxt; ! Move to the next block
538 0917     nxt = .nxt[d_l_uch];
539 0918     END;
540 0919
541 0920 RETURN;
542 0921 END;

```

```

                                007C 0000 INSERT_DEVICE:
                                .WORD Save R2,R3,R4,R5,R6
                                55      04 AC 7D 00002   MOVQ NEW, R5
                                54      08 BC D0 00006   MOVL @HEAD, NXT
                                44 A5    44 A4    08 13 0000A 1$: BEQL 2$
                                10 29 0000C   CMPC3 #16, 68(NXT), 68(R5)
                                09 1B 00012   BLEQU 3$
                                : 0877
                                : 0909
                                : 0896
                                : 0901
                                : 0909
                                :

```

SHODEV
V04-000

F 7
16-Sep-1984 01:32:33 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:09:25 [CLIUTL.SRC]SHODEV.B32;1

Page 21
(10)

04	66	04	AC	D0	00014	2\$:	MOVL	NEW, (PRV)	:	0912
	BC		54	D0	00018		MOVL	NXT, @NEW	:	0913
				04	0001C		RET		:	0911
	56		54	D0	0001D	3\$:	MOVL	NXT, PRV	:	0916
	54		64	D0	00020		MOVL	(NXT), NXT	:	0917
			E5	11	00023		BRB	1\$:	0898

; Routine Size: 37 bytes, Routine Base: \$CODE\$ + 02B2

```

544 0922 1 ROUTINE parse_device (device_desc, node, device, unit, flags) =
545 0923 BEGIN
546 0924
547 0925 ---
548 0926
549 0927 This routine takes the device specified in DEVICE_DESC and returns
550 0928 a device string and a unit number. The method used is to first translate
551 0929 the passed string up to 10 times, and then to parse the final, resultant
552 0930 string, breaking it into a unit number (possibly) and a DDB part.
553 0931
554 0932 Inputs
555 0933 DEVICE_DESC - address of a descriptor holding the actual string specified
556 0934
557 0935 Outputs
558 0936 NODE - address of ASCII to hold the node name or allocation class
559 0937 DEVICE - address of ASCII string to hold the DDB name
560 0938 UNIT - address of a longword to hold the unit number
561 0939
562 0940 ---
563 0941
564 0942 MAP
565 0943 node : REF VECTOR[.BYTE],
566 0944 device : REF VECTOR[.BYTE],
567 0945 flags : REF $BLOCK,
568 0946 device_desc : REF $BLOCK;
569 0947
570 0948 LOCAL
571 0949 status,
572 0950 exp,
573 0951 ptr,
574 0952 temp,
575 0953 temp_unit,
576 0954 in_desc : VECTOR[2],
577 0955 out_desc : VECTOR[2],
578 0956 in_buff : VECTOR[log$C_namlength, .BYTE],
579 0957 out_buff : VECTOR[log$C_namlength, .BYTE];
580 0958
581 0959
582 0960
583 0961 Transfer the initial string to IN_DESC, and set up the descriptors for the
584 0962 iterative translations.
585 0963
586 0964 in_desc[0] = .device_desc[dsc$w_length];           ! Use passed length
587 0965 in_desc[1] = in_buff;                             ! point to local buffer
588 0966 out_desc[0] = log$C_namlength;                     ! Set up the result descriptor
589 0967 out_desc[1] = out_buff;                             ! to be max size
590 0968 CH$MOVE(.device_desc[dsc$w_length],                ! Move original string to
591 0969 .device_desc[dsc$a_pointer],                       ! the local buffer.
592 0970 in_buff);
593 0971
594 0972
595 0973 Transfer the device name, up to 10 times.
596 0974
597 0975 INCR index FROM 1 TO 10 DO
598 0976 BEGIN
599 0977 ptr = CH$FIND (CH(.in_desc[0], .in_desc[1], ':'));
600 0978 IF NOT (CH$FAIC(.ptr))

```



```

601      0979      THEN in_desc[0] = .ptr - .in_desc[1];      ! Get rid of colons +...
602      0980
603      0981      status = STRNLOG(LOGNAM = in_desc,      ! Try to translate
604      0982      RSLBUF = out_desc,
605      0983      RSLLEN = out_desc);
606      0984
607      0985      IF NOT status      ! If an error, then exit
608      0986      THEN RETURN status;
609      0987
610      0988      temp = 0;      ! Before continuing with the
611      0989      IF CHSRCHAR(.out_desc[1]) EQL 'X'IB'      ! translation, see if this is
612      0990      AND CHSRCHAR(.out_desc[1] + 1) EQL 0      ! a process-permanent file.
613      0991      AND .out_desc[0] = 4 GTR 0      ! If so, then strip the first
614      0992      THEN temp = 4      ! four bytes (PPF info).
615      0993      ELSE IF CHSRCHAR(.out_desc[1]) EQL '_'      ! Also check and remove one or
616      0994      THEN      ! two underscores.
617      0995      BEGIN
618      0996      temp = 1;
619      0997      IF CHSRCHAR(.out_desc[1] + 1) EQL '_'
620      0998      THEN temp = 2;
621      0999      END;
622      1000      IF .temp NEQ 0      ! If there were any characters
623      1001      THEN      ! to remove, then do so now.
624      1002      BEGIN
625      1003      CHSMOVE(.out_desc[0] - .temp,
626      1004      .out_desc[1] + .temp,
627      1005      .out_desc[1]);
628      1006      out_desc[0] = .out_desc[0] - .temp;
629      1007      END;
630      1008
631      1009      ptr = .in_desc[1];      ! Rearrange the pointers
632      1010      in_desc[0] = .out_desc[0];      ! so that we continue to
633      1011      in_desc[1] = .out_desc[1];      ! translate.
634      1012      IF .status EQL $$$_NOTRAN      ! If no more translation,
635      1013      THEN EXITLOOP;      ! then go parse the string
636      1014      out_desc[0] = log$c_namlength;
637      1015      out_desc[1] = .ptr;
638      1016      END;
639      1017
640      1018      ! See if there is an SCS nodename or an allocation class on the front of the name.
641      1019      ! If an SCS nodename, remove it and place the SCS node into NODE as an ASCII string.
642      1020      ! If an allocation class, remove it and place it in NODE[1:4] as an integer, and
643      1021      ! set a flag so that we will know that it is an integer.
644      1022
645      1023      temp = CHSFIND(CH(.in_desc[0], .in_desc[1], '$'));
646      1024      IF NOT CHSFAIL7(temp)      ! If there is an SCS node,
647      1025      THEN
648      1026      BEGIN
649      1027      IF .temp EQL .in_desc[1]      ! Leading '$' should be an allocation name
650      1028      THEN
651      1029      BEGIN
652      1030      LOCAL
653      1031      alld : VECTOR [2, LONG];      ! Descriptor for allocation class string
654      1032      in_desc[0] = .in_desc[0] - 1;      ! Remove the first '$' from the front
655      1033      in_desc[1] = .in_desc[1] + 1;
656      1034      temp = CHSFIND_CR(.in_desc[0],      ! Find the second '$' in the name, the one
657      1035      .in_desc[1], '$');      ! that separates allocation class from device

```

```

658 1036 4 IF CH$FAIL(.temp) | If there is a '$' on the front, we'd better have
659 1037 4 THEN RETURN SSS_IVDEVNAM; | another one
660 1038 4 alld[0] = .temp - .in_desc[1]; | Calculate length of allocation class
661 1039 4 alld[1] = .in_desc[1]; | And fill in the address
662 1040 4 IF NOT OTSS$CVT-TIL(alld, node[0]) | Convert to an integer and store in node
663 1041 4 THEN RETURN SSS_IVDEVNAM; | Couldn't convert it, can't be valid
664 1042 4 flags[devi$var[ocls] = 1; | Let them know it is an allocation class name
665 1043 4 in_desc[0] = .in_desc[0] - .alld[0] - 1; | Calculate new length and
666 1044 4 in_desc[1] = .in_desc[1] + .alld[0] + 1; | new position of string
667 1045 4 END
668 1046 3 ELSE
669 1047 4 BEGIN
670 1048 4 node[0] = .temp - .in_desc[1]; | Calculate NODE count
671 1049 4 CH$MOVE(.node[0], | Put the node in NODE,
672 1050 4 .in_desc[1], | without the '$'
673 1051 4 node[1]);
674 1052 4 in_desc[0] = .in_desc[0] - .node[0] - 1; | Calculate new length and
675 1053 4 in_desc[1] = .in_desc[1] + .node[0] + 1; | new position of string
676 1054 4 END;
677 1055 4 END;
678 1056 2
679 1057 2
680 1058 2 | At this point we should have a physical device name, or else some fragment
681 1059 2 | of a device name. This fragment needs to be parsed into a unit number and
682 1060 2 | a device type. The simplest approach to take is to go backward, from the
683 1061 2 | end of the string, and locate the first non-numerical character.
684 1062 2
685 1063 2 IF .in_desc[0] EQL 0 | If no more, then
686 1064 2 THEN RETURN 1; | stop now.
687 1065 2
688 1066 2 exp = 1;
689 1067 2 temp = -1;
690 1068 2 temp_unit = 0;
691 1069 2 DECR index FROM .in_desc[0] - 1 TO 0 DO
692 1070 2 BEGIN
693 1071 2 LOCAL
694 1072 2 char : BYTE; | Temp character
695 1073 2
696 1074 2 char = CH$RCHAR(.in_desc[1] + .index); | Get a character
697 1075 2 IF .char GTR '9' | See if it's a number
698 1076 2 OR .char LSS '0'
699 1077 2 THEN (temp = .index; EXITLOOP) | If not, then exit
700 1078 2 ELSE | If a number, add it
701 1079 2 BEGIN | in to the unit number
702 1080 2 temp_unit = .temp_unit + ((.char - '0') * .exp);
703 1081 2 exp = .exp * 10;
704 1082 2 END;
705 1083 2 END;
706 1084 2
707 1085 2 IF .temp EQL -1 | If nothing but numbers,
708 1086 2 THEN RETURN SSS_IVDEVNAM; | return invalid device name.
709 1087 2
710 1088 2
711 1089 2 | TEMP points to the character just before the unit number. Copy
712 1090 2 | the string up to TEMP to the ASCII string, DEVICE.
713 1091 2
714 1092 2 CH$MOVE(.temp + 1, | Copy the device part

```

```

: 715      1093      2      .in_desc[1],      ! of the translated name
: 716      1094      2      device[1]);      ! to DEVICE
: 717      1095      2      device[0] = .temp + 1;      ! and put in the count
: 718      1096      2
: 719      1097      2
: 720      1098      2      ! If TEMP is not positioned at the last character of the string,
: 721      1099      2      ! then there is a unit number to get.
: 722      1100      2
: 723      1101      2      IF .temp+1 LSS .in_desc[0]
: 724      1102      2      THEN .unit = .temp_unit;
: 725      1103      2
: 726      1104      2      RETURN 1;
: 727      1105      1      END;

```

.EXTRN SYS\$TRNLOG

				03FC 0000	PARSE_DEVICE:		
		SE	FF68	CE 9E 00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9	0922
		50	04	AC D0 00007	MOVAB	-152(SP), SP	
		FB AD		60 3C 0000B	MOVL	DEVICE_DESC, R0	0964
		FC AD	48	AE 9E 0000F	MOVZWL	(R0), IN_DESC	
		F0 AD	40	8F 9A 00014	MOVZBL	#6, OUT_DESC	0965
		F4 AD	08	AE 9E 00019	MOVAB	OUT_BUFF, OUT_DESC+4	0966
48	AE	04	80	60 28 0001E	MOVAB	OUT_BUFF, OUT_DESC+4	0967
			58	01 D0 00024	MOVZBL	(R0), @4(R0), IN_BUFF	0968
FC	BD	FB	AD	3A 3A 00027	MOVL	#1, INDEX	0975
				02 12 0002D	LOCC	#58, IN_DESC, @IN_DESC+4	0977
				51 D4 0002F	BNEQ	2\$	
			56	51 D0 00031	CLRL	R1	
				06 13 00034	MOVL	R1, PTR	
FB	AD		56	FC AD C3 00036	BEQL	3\$	0978
				7E 7C 0003C	SUBL3	IN_DESC+4, PTR, IN_DESC	0979
				7E D4 0003E	CLRL	-(SP)	0983
				F0 AD 9F 00040	CLRL	-(SP)	
				F0 AD 9F 00043	PUSHAB	OUT_DESC	
				F8 AD 9F 00046	PUSHAB	OUT_DESC	
		00000000G	00	F8 AD 9F 00046	PUSHAB	IN_DESC	
			59	06 FB 00049	CALLS	#6, SYS\$TRNLOG	
			04	50 D0 00050	MOVL	R0, STATUS	
			50	59 E8 00053	BLBS	STATUS, 4\$	0984
				59 D0 00056	MOVL	STATUS, R0	0985
				04 00059	RET		
			50	57 D4 0005A	CLRL	TEMP	0987
			18	F4 AD D0 0005C	MOVL	OUT_DESC+4, R0	0988
				60 91 00060	CMPB	(R0), #27	
				10 12 00063	BNEQ	5\$	
				01 A0 95 00065	TSTB	1(R0)	0989
				0B 12 00068	BNEQ	5\$	
			04	F0 AD D1 0006A	CMPB	OUT_DESC, #4	0990
				05 15 0006E	BLEQ	5\$	
			57	04 D0 00070	MOVL	#4, TEMP	0991
				13 11 00073	BRB	6\$	
		SF	8F	60 91 00075	CMPB	(R0), #95	0992
				0D 12 00079	BNEQ	6\$	
			57	01 D0 0007B	MOVL	#1, TEMP	0995

		5F	8F	01	A0	91	0007E		CMPB	1(R0), #95	0996	
					03	12	00083		BNEQ	6\$	0997	
			57		02	D0	00085		MOVL	#2, TEMP	0999	
					57	D5	00088	6\$:	TSTL	TEMP	1002	
					0E	13	0008A		BEQL	7\$	1004	
	51	FO	AD		57	C3	0008C		SUBL3	TEMP, OUT_DESC, R1	1005	
	60		6740		51	28	00091		MOV3	R1, (TEMP[R0]), (R0)	1008	
		FO	AD		57	C2	00096		SUBL2	TEMP, OUT_DESC	1009	
			56	FC	AD	D0	0009A	7\$:	MOVL	IN_DESC+4, PTR	1011	
		F8	AD	FO	AD	7D	0009E		MOVQ	OUT_DESC, IN_DESC	1013	
		00000629	8F		59	D1	000A3		CMPL	STATUS, #1577	1014	
					0F	13	000AA		BEQL	8\$	0975	
			FO	40	8F	9A	000AC		MOVZBL	#64, OUT_DESC	1023	
			F4		56	D0	000B1		MOVL	PTR, OUT_DESC+4	1024	
FF6C			01		0A	F1	000B5		ACBL	#10, #1, INDEX, 1\$	1027	
	FC	58			24	3A	000BB	8\$:	LOCC	#36, IN_DESC, @IN_DESC+4	1032	
		BD	F8	AD	02	12	000C1		BNEQ	9\$	1033	
					51	D4	000C3		CLRL	R1	1034	
					51	D0	000C5	9\$:	MOVL	R1, TEMP	1036	
					7B	13	000C8		BEQL	15\$	1038	
					AC	D0	000CA		MOVL	NODE, R6	1039	
			FC	AD	57	D1	000CE		CMPL	TEMP, IN_DESC+4	1040	
					4A	12	000D2		BNEQ	13\$	1042	
					F8	AD	D7	000D4	DECL	IN_DESC	1043	
					F8	AD	D6	000D7	INCL	IN_DESC+4	1044	
	FC	BD	F8	AD	24	3A	000DA		LOCC	#38, IN_DESC, @IN_DESC+4	1048	
					02	12	000E0		BNEQ	10\$	1049	
					51	D4	000E2		CLRL	R1	1051	
					51	D0	000E4	10\$:	MOVL	R1, TEMP	1052	
					03	12	000E7		BNEQ	12\$	1053	
					009B	31	000E9	11\$:	BRW	21\$	1058	
					AD	C3	000EC	12\$:	SUBL3	IN_DESC+4, TEMP, ALLD	1063	
	6E		57	FC	AD	D0	000F1		MOVL	IN_DESC+4, ALLD+4	1064	
		04	AE	FC	56	DD	000F6		PUSHL	R6	1066	
					04	AE	9F	000F8	PUSHAB	ALLD	1067	
		00000000G	00		02	FB	000FB		CALLS	#2, OTSSCVT_TI_L	1068	
			E4		50	E9	00102		BLBC	R0, 11\$	1069	
			50	14	AC	D0	00105		MOVL	FLAGS, R0	1070	
			01		10	88	00109		BISB2	#16, 1(R0)	1071	
			F8	AD	6E	C3	0010D		SUBL3	ALLD, IN_DESC, R0	1072	
	50		AD	FF	A0	9E	00112		MOVAB	-1(R0), IN_DESC	1073	
			AD		6E	C1	00117		ADDL3	ALLD, IN_DESC+4, R0	1074	
					22	11	0011C		BRB	14\$	1075	
					AD	83	0011E	13\$:	SUBB3	IN_DESC+4, TEMP, (R6)	1076	
			57	FC	66	9A	00123		MOVZBL	(R6), R0	1077	
			50		50	28	00126		MOV3	R0, @IN_DESC+4, 1(R6)	1078	
	01	A6	FC	BD	66	9A	0012C		MOVZBL	(R6), R0	1079	
					50	C3	0012F		SUBL3	R0, IN_DESC, R0	1080	
			50		AD	9E	00134		MOVAB	-1(R0), IN_DESC	1081	
			F8	AD	66	9A	00139		MOVZBL	(R6), R0	1082	
					AD	C0	0013C		ADDL2	IN_DESC+4, R0	1083	
					01	A0	9E	00140	14\$:	MOVAB	1(R0), IN_DESC+4	1084
					F8	AD	D0	00145	15\$:	MOVL	IN_DESC, R6	1085
					5A	13	00149		BEQL	23\$	1086	
					01	D0	0014B		MOVL	#1, EXP	1087	
					01	CE	0014E		MNEGL	#1, TEMP	1088	
			52		59	D4	00151		CLRL	TEMP_UNIT	1089	
			57								1090	

50			58	D0	00153	MOVL	R8, INDEX	:	1074
			23	11	00156	BRB	19\$:	
51			0	90	00158	16\$:	MOV B @IN_DESC+4[INDEX], CHAR	:	
39			51	91	0015D		CMP B CHAR, #57	:	1075
			05	1A	00160		BGTRU 17\$:	
30			51	91	00162		CMP B CHAR, #48	:	1076
			05	1E	00165		BGEQU 18\$:	
57			50	D0	00167	17\$:	MOVL INDEX, TEMP	:	1077
			12	11	0016A		BRB 20\$:	
51			51	9A	0016C	18\$:	MOVZBL CHAR, R1	:	1080
51			30	C2	0016F		SUBL2 #48, R1	:	
51			52	C4	00172		MULL2 EXP, R1	:	
59			51	C0	00175		ADDL2 R1, TEMP_UNIT	:	
52			0A	C4	00178		MULL2 #10, EXP	:	1081
DA			50	F4	0017B	19\$:	SOBGEQ INDEX, 16\$:	1069
FFFFFFF			8F	D1	0017E	20\$:	CMPL TEMP, #-1	:	1085
			06	12	00185		BNEQ 22\$:	
50			8F	3C	00187	21\$:	MOVZWL #324, R0	:	1086
				04	0018C		RET	:	
			57	D6	0018D	22\$:	INCL R7	:	1092
			AC	D0	0018F		MOVL DEVICE, R6	:	1094
01	A6		BD	28	00193		MOV C3 R7, @IN_DESC+4, 1(R6)	:	
			66	90	00199		MOV B R7, (R6)	:	1095
			58	D1	0019C		CMPL R7, R8	:	1101
			04	18	0019F		BGEQ 23\$:	
			10	BC	001A1		MOVL TEMP_UNIT, @UNIT	:	1102
			50	D0	001A5	23\$:	MOVL #1, R0	:	1104
			01	D0	001A8		RET	:	1105
				04	001A8			:	

: Routine Size: 425 bytes, Routine Base: \$CODE\$ + 02D7

: 729 1106 1 END
: 730 1107 0 ELUDOM

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes
\$OWNS	28	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$PLITS	156	NOVEC, NOWRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$CODES	1152	NOVEC, NOWRT, RD, EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)

Library Statistics

File	Symbols		Pages Mapped	Processing Time
	Total	Loaded Percent		
_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	37 0	1000	00:01.7

COMMAND QUALIFIERS

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

: Size: 1152 code + 184 data bytes
: Run Time: 00:33.4
: Elapsed Time: 01:42.9
: Lines/CPU Min: 1989
: Lexemes/CPU-Min: 44590
: Memory Used: 210 pages
: Compilation Complete

0055 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

The image displays a grid of 140 terminal windows arranged in 10 rows and 14 columns. Each window contains text-based data, likely system logs or diagnostic information. Several windows are highlighted with larger, bold text:

- Row 2, Column 10: SHODEVPT LIS
- Row 7, Column 14: SHODEVUTL LIS
- Row 8, Column 5: SHODEV LIS
- Row 10, Column 7: SHODEVCLU LIS

The text in the other windows is smaller and less legible, but appears to be structured data or log entries.