

DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDD	DDD	EEE	UUU	UUU	GGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG

```

DDDDDDDD  BBBB8888  GGGGGGGG  SSSSSSSS  SSSSSSSS  VV      VV
DDDDDDDD  BBBB8888  GGGGGGGG  SSSSSSSS  SSSSSSSS  VV      VV
DD      DD  BB      BB  GG      SS      SS      VV      VV
DD      DD  BB      BB  GG      SS      SS      VV      VV
DD      DD  BB      BB  GG      SS      SS      VV      VV
DD      DD  BB      BB  GG      SS      SS      VV      VV
DD      DD  BBBB8888  GG      SSSSSS  SSSSSS  VV      VV
DD      DD  BBBB8888  GG      SSSSSS  SSSSSS  VV      VV
DD      DD  BB      BB  GG  GGGGGG  SS      SS  VV      VV
DD      DD  BB      BB  GG  GGGGGG  SS      SS  VV      VV
DD      DD  BB      BB  GG      GG      SS      SS  VV      VV
DD      DD  BB      BB  GG      GG      SS      SS  VV      VV
DDDDDDDD  BBBB8888  GGGGGG  SSSSSSSS  SSSSSSSS  VV      VV
DDDDDDDD  BBBB8888  GGGGGG  SSSSSSSS  SSSSSSSS  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 DBGSSV ( IDENT = 'V04-000' ) =
2 0002 1 BEGIN
3 0003 1
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 FACILITY:      DEBUG
29 0029 1
30 0030 1 **
31 0031 1 FUNCTIONAL DESCRIPTION:
32 0032 1     Contains SHOW MODULE and SHOW TYPE code.
33 0033 1
34 0034 1 Version:      1.0
35 0035 1
36 0036 1 History:
37 0037 1     Author:
38 0038 1         Carol Peters, 21 Sep 1976: Version 01
39 0039 1
40 0040 1     Most of the code was eliminated for version 4 of DEBUG, with
41 0041 1     only the SHOW MODULE and SHOW TYPE routines remaining.
42 0042 1         - R. Title 28 Apr 1983
43 0043 1
44 0044 1     Add support for PACKED type
45 0045 1         - W. Carrell 12-Jul-83
46 0046 1
47 0047 1     Add support for DATE_TIME type
48 0048 1         - W. Carrell 18-Jul-83
49 0049 1 --
```

```

51      0050 1 ! TABLE OF CONTENTS:
52      0051 1 !
53      0052 1 FORWARD ROUTINE
54      0053 1     DBG$OUT_NUM_VAL : NOVALUE,
55      0054 1     DBG$READ_ACCESS,           ! PROBE read accessibility.
56      0055 1     DBG$SHOW_MODULE : NOVALUE, ! List off the module chain.
57      0056 1     DBG$SHOW_TYPE : NOVALUE;   ! Print out the current TYPE.
58      0057 1
59      0058 1 !
60      0059 1 ! REQUIRE FILES:
61      0060 1 !
62      0061 1
63      0062 1 REQUIRE 'SRC$:DBGPROLOG.REQ';
64      0196 1 LIBRARY 'LIB$:DBGGEN.L32';
65      0197 1
66      0198 1 MACRO
67      M 0199 1     IF_SIGNAL (code) =
68      M 0200 1         IF .signal_flag NEQ 0
69      M 0201 1         THEN
70      M 0202 1             BEGIN
71      M 0203 1                 IF NOT
72      M 0204 1                     ( IF %LENGTH GTR 1
73      M 0205 1                         THEN dbg$nout_info (code, %REMAINING)
74      M 0206 1                             ELSE dbg$nout_info (code))
75      M 0207 1                 THEN
76      M 0208 1                     BEGIN
77      M 0209 1                         .signal_flag = (IF %LENGTH GTR 1
78      M 0210 1                             THEN
79      M 0211 1                                 dbg$make_arg_vect (code, %REMAINING)
80      M 0212 1                             ELSE
81      M 0213 1                                 dbg$make_arg_vect (code));
82      M 0214 1                         RETURN sts$k_severe;
83      M 0215 1                     END
84      M 0216 1                 END
85      M 0217 1             ELSE
86      M 0218 1                 BEGIN
87      M 0219 1                     IF %LENGTH GTR 1
88      M 0220 1                         THEN
89      M 0221 1                             SIGNAL (code, %REMAINING)
90      M 0222 1                         ELSE
91      M 0223 1                             SIGNAL (code)
92      M 0224 1                         END %;
93      M 0225 1             END %;
94      M 0226 1 MACRO
95      M 0227 1     SET_FLAG (param_num) =
96      M 0228 1         LOCAL
97      M 0229 1             signal_flag;
98      M 0230 1
99      M 0231 1         signal_flag = (IF actualcount () GTR param_num
100     M 0232 1             THEN
101     M 0233 1                 actualparameter (actualcount())
102     M 0234 1             ELSE
103     M 0235 1                 0) %;
104     M 0236 1

```

```

: 106      0237      :
: 107      0238      :  : EQUATED SYMBOLS:
: 108      0239      :
: 109      0240      :
: 110      0241      : EXTERNAL ROUTINE
: 111      0242      :     dbg$flushbuf: NOVALUE,      ! Initializes new print line.
: 112      0243      :     dbg$free_mem_left,         ! Longwords remaining in free storage.
: 113      0244      :     dbg$language,             ! Report on a given language name.
: 114      0245      :     dbg$ngget_radix,          ! Obtain radix
: 115      0246      :     dbg$nmak_e_arg_vect,
: 116      0247      :     dbg$newline: NOVALUE,      ! Outputs print buffer to output device
: 117      0248      :     dbg$nout_info,
: 118      0249      :     dbg$print: NOVALUE,        ! Formats print buffer for output
: 119      0250      :     dbg$sta_symname: NOVALUE,
: 120      0251      :     for$cnv_out_i;            ! Converts integer to ascii string.
: 121      0252      :
: 122      0253      : EXTERNAL
: 123      0254      :     dbg$gw_dfltleng : WORD,      ! The length specified in a SET TYPE statement.
: 124      0255      :     dbg$gw_gblngth  : WORD,      ! the length given in a SET TYPE/OVERRIDE ASCII:LENGTH comma
: 125      0256      :     dbg$gl_gbltyp,  ! the type given in a SET TYPE/OVERRIDE command.
: 126      0257      :     dbg$gl_dflttyp, ! the type specified in a SET TYPE statement.
: 127      0258      :     rst$start_addr: REF rst$entry; ! Pointer to the module chain (MC).
```

```

129 0259 1 GLOBAL ROUTINE DBG$OUT_NUM_VAL (VALUE) : NOVALUE =
130 0260 1
131 0261 1 Functional Description:
132 0262 1     Write out the given value.
133 0263 1
134 0264 1 Inputs:
135 0265 1     VALUE - the actual value we are to write out.
136 0266 1
137 0267 1 Outputs:
138 0268 1     The (numeric) character representation
139 0269 1     of the value is encoded into the output buffer.
140 0270 1
141 0271 1 Implicit Outputs:
142 0272 1     None.
143 0273 1
144 0274 1 Routine Value:
145 0275 1     NOVALUE.
146 0276 1 ---
147 0277 2 BEGIN
148 0278 2
149 0279 2 ! We want to look at the passed-in value
150 0280 2 ! as a byte vector only to see if we may
151 0281 2 ! need to put in a leading 0 for hex output.
152 0282 2
153 0283 2 MAP
154 0284 2     value : vector[ %UPVAL, byte ];
155 0285 2
156 0286 2 LOCAL
157 0287 2     use_radix;
158 0288 2
159 0289 2 OWN
160 0290 2     format : VECTOR[4,BYTE] INITIAL(BYTE(%ASCIC '!??'));
161 0291 2
162 0292 2     use_radix = dbg$nget_radix();
163 0293 2
164 0294 2 ! Now just build the required 2-character format
165 0295 2 ! string descriptor, based upon the current
166 0296 2 ! setting of the radix.
167 0297 2
168 0298 2     format[2] = (SELECTONE .use_radix OF
169 0299 2         SET
170 0300 2             [dbg$k_binary]:           'B';
171 0301 2             [dbg$k_octal]:           'O';
172 0302 2             [dbg$k_decimal]:         'S';
173 0303 2             [dbg$k_hex]:             'X';
174 0304 2         YES
175 0305 2     );
176 0306 2     format[3] = 'L';
177 0307 2
178 0308 2 ! Check for hex output which begins with A-F
179 0309 2
180 0310 2 IF .use_radix EQL dbg$k_hex
181 0311 2 THEN
182 0312 2
183 0313 2     ! Perform the check by extracting the upper BYTE of
184 0314 2     ! what we will later output, comparing that byte with
185 0315 2     ! the max hex number we will print out 'unadorned',

```

```

: 186
: 187
: 188
: 189
: 190
: 191
: 192
: 193
: 194
: 195
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325

```

```

! and then inserting a leading 0 if the check fails.
IF .value[3] GTRA %X'9F'
THEN
    dbg$print(UPLIT(%ASCIC '0'));
! Output the value and we're done.
dbg$print(format,.value);
END;

```

```

.TITLE DBGSSV
.IDENT \V04-000\

.PSECT DBG$PLIT,NOWRT, SHR, PIC,0
00 00 30 01 00000 P.AAA: .ASCII <1>\0\<0><0> ;
.PSECT DBG$OWN,NOEXE, PIC,2
3F 3F 21 03 00000 FORMAT: .ASCII <3>\!??\ ;

.EXTRN DBG$FLUSHBUF, DBG$FREE_MEM_LEFT
.EXTRN DBG$LANGUAGE, DBG$NGET_RADIX
.EXTRN DBG$NMAKE_ARG_VECT
.EXTRN DBG$NEWLINE, DBG$NOUT_INFO
.EXTRN DBG$PRINT, DBG$STA_SYMNAME
.EXTRN FOR$CNV_OUT_I, DBG$GW_DFLTLENG
.EXTRN DBG$GW_GBLLENGLTH
.EXTRN DBG$GL_GBLTYP, DBG$GL_DFLTTP
.EXTRN RST$START_ADDR

.PSECT DBG$CODE,NOWRT, SHR, PIC,0
.ENTRY DBG$OUT_NUM_VAL, Save R2,R3 : 0259
MOVAB DBG$PRINT, R3
MOVAB FORMAT+2, R2
CALLS #0, DBG$NGET_RADIX : 0292
CMPL USE_RADIX, #2 : 0300
BNEQ 1$
MOVZBL #66, R1
BRB 5$
CMPL USE_RADIX, #8 : 0301
BNEQ 2$
MOVZBL #79, R1
BRB 5$
CMPL USE_RADIX, #10 : 0302
BNEQ 3$
MOVZBL #83, R1
BRB 5$
CMPL USE_RADIX, #16 : 0303
BNEQ 4$
MNEGL #1, R1
BRB 5$
MOVZBL #88, R1
MOVB R1, FORMAT+2 : 0298

```

DBGSSV
V04-000

01	A2	4C	8F	90	00049	MOVB	#76, FORMAT+3	:	0306
	10		50	D1	0004E	CMPL	USE_RADIX, #16	:	0310
			10	12	00051	BNEQ	6\$:	
9F	8F	07	AC	91	00053	CMPB	VALUE+3, #159	:	0318
			09	1B	00058	BLEQU	6\$:	
		00000000'	EF	9F	0005A	PUSHAB	P.AAA	:	0320
	63		01	FB	00060	CALLS	#1, DBG\$PRINT	:	
		04	AC	DD	00063	PUSHL	VALUE	:	0324
		FE	A2	9F	00066	PUSHAB	FORMAT	:	
	63		02	FB	00069	CALLS	#2, DBG\$PRINT	:	
			04	0006C		RET		:	0325

; Routine Size: 109 bytes. Routine Base: DBG\$CODE + 0000


```

197 0326 1 GLOBAL ROUTINE DBG$READ_ACCESS( ADDR, NUM_BYTES ) =
198 0327 1
199 0328 1 **
200 0329 1 Functional Description:
201 0330 1 Check that we have READ access for the number
202 0331 1 of bytes indicated, starting at the given address.
203 0332 1
204 0333 1 Note that this routine has been modified to use version 3 error
205 0334 1 reporting, when appropriate.
206 0335 1
207 0336 1 Inputs:
208 0337 1 ADDR -Byte address where desired data begins.
209 0338 1 NUM_BYTES -Number of bytes which we want to access.
210 0339 1 [message_vect] - Address of a longword to contain the address of a
211 0340 1 message argument vector
212 0341 1
213 0342 1 Implicit Inputs:
214 0343 1 The PROBER instruction insists on using PSL <prev>, whether
215 0344 1 we want it to or not. Until/unless DEBUG is changed to be a
216 0345 1 multi-mode tool, we will be OK as we are now. Then, however,
217 0346 1 we will have to put in substantially more code here to ensure
218 0347 1 that current and previous mode are the same.
219 0348 1
220 0349 1 Outputs:
221 0350 1 None.
222 0351 1
223 0352 1 Implicit Outputs:
224 0353 1 None.
225 0354 1
226 0355 1 Routine Value:
227 0356 1 TRUE - if access will be granted,
228 0357 1 If access is not possible routine signals and no return happens, if
229 0358 1 a version 2 routine has called. If a version 3 has called with the
230 0359 1 message_vect parameter, then this routine creates a message argument
231 0360 1 vector and returns error.
232 0361 1
233 0362 1 Side Effects:
234 0363 1 None.
235 0364 1
236 0365 1 --
237 0366 1
238 0367 2 BEGIN
239 0368 2 BUILTIN
240 0369 2 ACTUALCOUNT,
241 0370 2 ACTUALPARAMETER,
242 0371 2 PROBER;
243 0372 2
244 0373 2 ! Set up error handling for version 3 calls
245 0374 2
246 0375 2 set_flag (2);
247 0376 2
248 0377 2 ! Note that we must pass PROBER a zero-extended WORD
249 0378 2 address for the LENGTH, and a BYTE address containing
250 0379 2 the MODE field for the instruction.
251 0380 2 ! Later, this byte field should be built by extracting
252 0381 2 ! the proper 3 bits from the PSL.
253 0382 2

```

: 254 0383 2
: 255 0384 2
: 256 0385 2
: 257 0386 2
: 258 0387 2
: 259 0388 1

```
IF NOT PROBER( %REF(0), NUM_BYTES, .ADDR)
THEN
  IF_SIGNAL( dbg$_noaccessr, 1, .addr );      ! Output error message
RETURN TRUE;
END;
```

				0004 00000	.ENTRY	DBG\$READ_ACCESS, Save R2	: 0326
	02			6C 91 00002	CMPB	(AP), #2	: 0375
				09 1B 00005	BLEQU	1\$	
	50			6C 9A 00007	MOVZBL	(AP), R0	
	52			6C40 D0 0000A	MOVL	(AP)[R0], SIGNAL_FLAG	
				02 11 0000E	BRB	2\$	
				52 D4 00010	CLRL	SIGNAL_FLAG	
04	BC	08	AC	00 0C 00012	PROBER	#0, NUM_BYTES, @ADDR	: 0383
				44 12 00018	BNEQ	4\$	
				52 D5 0001A	TSTL	SIGNAL_FLAG	: 0385
				2E 13 0001C	BEQL	3\$	
			04	AC DD 0001E	PUSHL	ADDR	
				01 DD 00021	PUSHL	#1	
			00028228	8F DD 00023	PUSHL	#164392	
00000000G	00			03 FB 00029	CALLS	#3, DBG\$NOUT_INFO	
	2B			50 E8 00030	BLBS	R0, 4\$	
			04	AC DD 00033	PUSHL	ADDR	
				01 DD 00036	PUSHL	#1	
			00028228	8F DD 00038	PUSHL	#164392	
00000000G	00			03 FB 0003E	CALLS	#3, DBG\$NMAKE_ARG_VECT	
	62			50 D0 00045	MOVL	R0, (SIGNAL_FLAG)	
	50			04 D0 00048	MOVL	#4, R0	
				04 0004B	RET		
			04	AC DD 0004C	PUSHL	ADDR	: 3\$:
				01 DD 0004F	PUSHL	#1	
			00028228	8F DD 00051	PUSHL	#164392	
00000000G	00			03 FB 00057	CALLS	#3, LIB\$SIGNAL	
	50			01 D0 0005E	MOVL	#1, R0	: 0387
				04 00061	RET		: 0388

; Routine Size: 98 bytes, Routine Base: DBG\$CODE + 006D

```

261 0389 1 GLOBAL ROUTINE DBG$SHOW_MODULE: NOVALUE =
262 0390 1
263 0391 1 FUNCTION
264 0392 1 This routine prints out the SHOW MODULE display. It first scans all
265 0393 1 modules in the RST Module Chain to see if they are all written in the
266 0394 1 same language. If they are, the language name is omitted in the SHOW
267 0395 1 MODULE output. It then makes a second pass over the RST Module Chain
268 0396 1 to print out the appropriate information for each module in the chain.
269 0397 1
270 0398 1 Since the language code may not be set in the Module RST Entry for a
271 0399 1 module which is not SET, the language code is always retrieved from
272 0400 1 the Module Begin DST Record for each module.
273 0401 1
274 0402 1 INPUTS
275 0403 1 NONE
276 0404 1
277 0405 1 OUTPUTS
278 0406 1 NONE
279 0407 1
280 0408 1
281 0409 2 BEGIN
282 0410 2
283 0411 2 LOCAL
284 0412 2 DSTPTR: REF DST$RECORD, ; Pointer to Module Begin DST Record
285 0413 2 FIRST_TIME_FLAG, ; Flag set first time through loop
286 0414 2 LANG_CHANGE, ; Flag set if all languages are the same
287 0415 2 LANG_CODE, ; Language encoding for each module
288 0416 2 MODRSTPTR: REF RST$ENTRY, ; Pointer to current Module RST Entry
289 0417 2 NAMEPTR: REF VECTOR[BYTE], ; Pointer to ASCII language name
290 0418 2 NUMBER_MODULES; ; The number of modules in the program
291 0419 2
292 0420 2
293 0421 2
294 0422 2 ; Set up to do standard DEBUG I/O.
295 0423 2
296 0424 2 DBG$FLUSHBUF();
297 0425 2
298 0426 2
299 0427 2 ; If there is no RST Module Chain or if the only module on that chain is
300 0428 2 the "anonymous" module used for global symbols, then we signal a message
301 0429 2 and return immediately.
302 0430 2
303 0431 2 IF .RST$START_ADDR EQL 0
304 0432 2 THEN
305 0433 3 BEGIN
306 0434 3 SIGNAL(DBG$_NOLOCALS);
307 0435 3 RETURN;
308 0436 2 END;
309 0437 2
310 0438 2 IF .RST$START_ADDR[RST$_NEXTMODPTR] EQL 0
311 0439 2 THEN
312 0440 3 BEGIN
313 0441 3 SIGNAL(DBG$_NOLOCALS);
314 0442 3 RETURN;
315 0443 2 END;
316 0444 2
317 0445 2

```

```

318 0446 2
319 0447 2
320 0448 2
321 0449 2
322 0450 2
323 0451 2
324 0452 2
325 0453 2
326 0454 2
327 0455 2
328 0456 2
329 0457 2
330 0458 2
331 0459 2
332 0460 2
333 0461 2
334 0462 2
335 0463 2
336 0464 2
337 0465 2
338 0466 2
339 0467 2
340 0468 2
341 0469 2
342 0470 2
343 0471 2
344 0472 2
345 0473 2
346 0474 2
347 0475 2
348 0476 2
349 0477 2
350 0478 2
351 0479 2
352 0480 2
353 0481 2
354 0482 2
355 0483 2
356 0484 2
357 0485 2
358 0486 2
359 0487 2
360 0488 2
361 0489 2
362 0490 2
363 0491 2
364 0492 2
365 0493 2
366 0494 2
367 0495 2
368 0496 2
369 0497 2
370 0498 2
371 0499 2
372 0500 2
373 0501 2
374 0502 2

```

```

! See if all modules happen to be written in the same language. To do this
! we scan the whole RST Module Chain, checking the language codes. Note
! we always pick up the language code from the Module Begin DST Record as
! it may not be set in the Module RST Entry for a module which is not SET.

LANG_CHANGE = FALSE;
MODRSTPTR = .RST$START_ADDR[RST$L_NXTMODPTR];
FIRST_TIME_FLAG = TRUE;
WHILE .MODRSTPTR NEQ 0 DO
  BEGIN
    DSTPTR = .MODRSTPTR[RST$L_DSTPTR];
    IF .DSTPTR[DST$B_TYPE] NEQ DST$K_MODBEG THEN SIGNAL(DBG$ INV DSTREC);
    MODRSTPTR[RST$B_LANGUAGE] = .DSTPTR[DST$L_MODBEG_LANGUAGE];
    IF .FIRST_TIME_FLAG
    THEN
      BEGIN
        FIRST_TIME_FLAG = FALSE;
        LANG_CODE = .MODRSTPTR[RST$B_LANGUAGE];
      END

      ELSE IF .LANG_CODE NEQ .MODRSTPTR[RST$B_LANGUAGE]
      THEN
        BEGIN
          LANG_CHANGE = TRUE;
          EXIT[LOOP];
        END;

    MODRSTPTR = .MODRSTPTR[RST$L_NXTMODPTR];
  END;

! Print the column header lines for the SHOW MODULE display.
DBG$PRINT(UPLIT(%ASCIC 'module name symbols '));
IF .LANG_CHANGE THEN DBG$PRINT(UPLIT(%ASCIC ' language' ));
DBG$PRINT(UPLIT(%ASCIC ' size'));
DBG$NEWLINE();
DBG$NEWLINE();

! Loop through the RST Module Chain and for each module, print the relevant
! information. Note that we bypass the "anonymous" module at the start of
! the chain--it does not constitute a real module to the user.

MODRSTPTR = .RST$START_ADDR[RST$L_NXTMODPTR];
NUMBER_MODULES = 0;
WHILE .MODRSTPTR NEQ 0 DO
  BEGIN
    NUMBER_MODULES = .NUMBER_MODULES + 1;

    ! Abort the SHOW MODULE output if the user entered Control-Y DEBUG to
    ! stop the current command.
  $ABORT_ON_CONTROL_Y;

```

```

375 0503 ! Print out the standard information. Start by printing out the module
376 0504 ! name. Note that we special-case names longer than 31 characters.
377 0505 !
378 0506 DBG$FLUSHBUF();
379 0507 DBG$STA SYMNAME(.MODRSTPTR, NAMEPTR);
380 0508 IF .NAMEPTR[0] LEQ 31
381 0509 THEN
382 0510     DBG$PRINT(UPLIT(%ASCIC '!31AC '), .NAMEPTR)
383 0511
384 0512 ELSE
385 0513     BEGIN
386 0514     DBG$PRINT(UPLIT(%ASCIC '!63AC'), .NAMEPTR);
387 0515     DBG$NEWLINE();
388 0516     DBG$PRINT(UPLIT(%ASCIC '!32* '));
389 0517     END;
390 0518
391 0519 !
392 0520 ! Now print out whether or not the module has be SET, i.e. whether or
393 0521 ! not its RST has been built.
394 0522 !
395 0523 IF .MODRSTPTR[RST$V_MODSET]
396 0524 THEN
397 0525     DBG$PRINT(UPLIT(%ASCIC 'yes  '))
398 0526
399 0527 ELSE
400 0528     DBG$PRINT(UPLIT(%ASCIC 'no  '));
401 0529
402 0530 !
403 0531 ! Unless all modules are written in the same language, output the name
404 0532 ! of the language in which this module was written. Finally print the
405 0533 ! module's estimated RST size and output the buffer. Then link to the
406 0534 ! next module in the chain.
407 0535 !
408 0536 DSTPTR = .MODRSTPTR[RST$L_DSTPTR];
409 0537 MODRSTPTR[RST$B_LANGUAGE] = .DSTPTR[DST$L_MODBEG_LANGUAGE];
410 0538 IF .LANG_CHANGE
411 0539 THEN
412 0540     BEGIN
413 0541     IF .MODRSTPTR[RST$V_SHARE_IMAGE]
414 0542     THEN
415 0543         DBG$PRINT(UPLIT(%ASCIC ' Image  '))
416 0544     ELSE
417 0545         DBG$PRINT(UPLIT(%ASCIC ' !8AC'),
418 0546                 DBG$LANGUAGE(.MODRSTPTR[RST$B_LANGUAGE]));
419 0547     END;
420 0548
421 0549 DBG$PRINT(UPLIT(%ASCIC '!7SL'), .MODRSTPTR[RST$L_MODRSTSI2]);
422 0550 DBG$NEWLINE();
423 0551 MODRSTPTR = .MODRSTPTR[RST$L_NXTMODPTR];
424 0552 END;
425 0553
426 0554 !
427 0555 ! Print the summary information at the end of the SHOW MODULE output. This
428 0556 ! includes the total number of modules and the number of bytes of free
429 0557 ! memory left. The language name is included if all modules were written
430 0558 ! in the same language.
431 0559 !

```

```

: 432      0560      2      DBG$NEWLINE();
: 433      0561      2      DBG$PRINT(UPLIT(%ASCIC 'total '));
: 434      0562      2      IF NOT .LANG_CHANGE
: 435      0563      2      THEN
: 436      0564      2          DBG$PRINT(UPLIT(%ASCIC '!AC '), DBG$LANGUAGE(.LANG_CODE));
: 437      0565      2
: 438      0566      2      DBG$PRINT(UPLIT(%ASCIC 'modules: !SL. '), .NUMBER_MODULES);
: 439      0567      2      DBG$PRINT(UPLIT(%ASCIC '!_!_remaining size: !SL. '), 4*DBG$FREE_MEM_LEFT());
: 440      0568      2      DBG$NEWLINE();
: 441      0569      2
: 442      0570      2      END;

```

INFO#250 L1:0466
: Referenced LOCAL symbol LANG_CODE is probably not initialized

															.PSECT			DBG\$PLIT, NOWRT, SHR, PIC, 0		
20	20	20	65	6D	61	6E	20	65	6C	75	64	6F	6D	28	00004	P.AAB:	.ASCII	\(module name	symbols\	:
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	00013					:
															00022					:
															0002C		.ASCII	\ \<0><0><0>		:
			00	65	67	61	75	67	6E	61	6C	20	20	0A	00030	P.AAC:	.ASCII	<10>\ language\<0>		:
							65	7A	69	73	20	20	07	0003C	P.AAD:	.ASCII	<7>\ size\		:	
							00	20	43	41	31	33	21	06	00044	P.AAE:	.ASCII	<6>\!31AC \<0>		:
							00	00	43	41	33	36	21	05	0004C	P.AAF:	.ASCII	<5>\!63AC \<0><0>		:
							00	00	20	2A	32	33	21	05	00054	P.AAG:	.ASCII	<5>\!32* \<0><0>		:
			00	00	00	20	20	20	20	20	73	65	79	08	0005C	P.AAH:	.ASCII	<8>\yes \<0><0><0>		:
			00	00	00	20	20	20	20	20	20	6F	6E	08	00068	P.AAI:	.ASCII	<8>\no \<0><0><0>		:
			00	20	20	20	65	67	61	6D	49	20	20	0A	00074	P.AAJ:	.ASCII	<10>\ Image \<0>		:
							00	43	41	38	21	20	20	06	00080	P.AAK:	.ASCII	<6>\ !8AC \<0>		:
							00	00	00	4C	53	37	21	04	00088	P.AAL:	.ASCII	<4>\!7SL \<0><0><0>		:
							00	20	6C	61	74	6F	74	06	00090	P.AAM:	.ASCII	<6>\total \<0>		:
00	2E	4C	53	21	20	3A	73	65	6C	75	64	6F	6D	0D	000A0	P.AAO:	.ASCII	<13>\modules: !SL.\<0><0>		:
														00	000AF					:
20	67	6E	69	6E	69	61	6D	65	72	5F	21	5F	21	18	000B0	P.AAP:	.ASCII	<24>\!_!_remaining size: !SL.\<0><0>		:
			00	00	2E	4C	53	21	20	3A	65	7A	69	73	000BF					:
														00	000CB		.ASCII	<0>		:
															.EXTRN			DBG\$GV_CONTROL		
															.PSECT			DBG\$CODE, NOWRT, SHR, PIC, 0		
															.ENTRY			DBG\$SHOW MODULE, Save R2,R3,R4,R5,R6,R7,R8,-; 0389		
																		R9,R10,RT1		
																		MOVAB LIB\$SIGNAL, R11		
																		MOVAB RST\$START_ADDR, R10		
																		MOVAB DBG\$NEWLINE, R9		
																		MOVAB DBG\$PRINT, R8		
																		MOVAB P.AAB, R7		
																		SUBL2 #4, SP		
																		CALLS #0, DBG\$FLUSHBUF		
																		TSTL RST\$START_ADDR		
																		BEQL 1\$		
																		MOVL RST\$START_ADDR, R0		
																		TSTL 16(R0)		

0424
0431
0438

			0A	12	00039		BNEQ	2\$			
			8F	DD	0003B	1\$:	PUSHL	#163923			0441
	68		01	FB	00041		CALLS	#1, LIB\$SIGNAL			
				04	00044		RET				0440
			56	D4	00045	2\$:	CLRL	LANG CHANGE			0451
	50		6A	D0	00047		MOVL	RST\$START_ADDR, R0			0452
	52	10	A0	D0	0004A		MOVL	16(R0), MODRSTPTR			
	54		01	D0	0004E		MOVL	#1, FIRST_TIME_FLAG			0453
			52	D5	00051	3\$:	TSTL	MODRSTPTR			0454
			37	13	00053		BEQL	7\$			
	53	0C	A2	D0	00055		MOVL	12(MODRSTPTR), DSTPTR			0456
	BC	8F	01	A3	91	00059	CMPB	1(DSTPTR), #188			0457
			09	13	0005E		BEQL	4\$			
			8F	DD	00060		PUSHL	#164650			
	68		01	FB	00066		CALLS	#1, LIB\$SIGNAL			
	29		A3	90	00069	4\$:	MOVW	3(DSTPTR), 41(MODRSTPTR)			0458
			54	E9	0006E		BLBC	FIRST_TIME_FLAG, 5\$			0459
			54	D4	00071		CLRL	FIRST_TIME_FLAG			0462
	55	29	A2	9A	00073		MOVZBL	41(MODRSTPTR), LANG_CODE			0463
			0D	11	00077		BRB	6\$			0459
55		29	A2	00	ED	00079	5\$:	CMPZV	#0, #8, 41(MODRSTPTR), LANG_CODE		0466
			05	13	0007F		BEQL	6\$			
	56		01	D0	00081		MOVL	#1, LANG_CHANGE			0469
			06	11	00084		BRB	7\$			0468
	52	10	A2	D0	00086	6\$:	MOVL	16(MODRSTPTR), MODRSTPTR			0473
			C5	11	0008A		BRB	3\$			0454
			57	DD	0008C	7\$:	PUSHL	R7			0479
	68		01	FB	0008E		CALLS	#1, DBG\$PRINT			
	06		56	E9	00091		BLBC	LANG CHANGE, 8\$			0480
		2C	A7	9F	00094		PUSHAB	P.AAC			
	68		01	FB	00097		CALLS	#1, DBG\$PRINT			
		38	A7	9F	0009A	8\$:	PUSHAB	P.AAD			0481
	68		01	FB	0009D		CALLS	#1, DBG\$PRINT			
	69		00	FB	000A0		CALLS	#0, DBG\$NEWLINE			0482
	69		00	FB	000A3		CALLS	#0, DBG\$NEWLINE			0483
	50		6A	D0	000A6		MOVL	RST\$START_ADDR, R0			0490
	52	10	A0	D0	000A9		MOVL	16(R0), MODRSTPTR			
			54	D4	000AD		CLRL	NUMBER MODULES			0491
			52	D5	000AF	9\$:	TSTL	MODRSTPTR			0492
			03	12	000B1		BNEQ	10\$			
			0095	31	000B3		BRW	18\$			
			54	D6	000B6	10\$:	INCL	NUMBER MODULES			0494
09	00000000G	00	01	E1	000B8		BBC	#1, DBG\$GV_CONTROL+1, 11\$			
			8F	DD	000C0		PUSHL	#164072			
			01	FB	000C6		CALLS	#1, LIB\$SIGNAL			
	00000000G	00	00	FB	000C9	11\$:	CALLS	#0, DBG\$FLUSHBUF			0506
			8F	BB	000D0		PUSHR	#*M<R2, SP>			0507
	00000000G	00	02	FB	000D4		CALLS	#2, DBG\$STA SYMNAME			
		1F	00	BE	91	000DB	CMPB	@NAMEPTR, #31			0508
			0A	1A	000DF		BGTRU	12\$			
			6E	DD	000E1		PUSHL	NAMEPTR			0510
		40	A7	9F	000E3		PUSHAB	P.AAE			
	68		02	FB	000E6		CALLS	#2, DBG\$PRINT			
			11	11	000E9		BRB	13\$			
			6E	DD	000EB	12\$:	PUSHL	NAMEPTR			0514
		48	A7	9F	000ED		PUSHAB	P.AAF			
	68		02	FB	000F0		CALLS	#2, DBG\$PRINT			

	69		00	FB	000F3		CALLS	#0, DBG\$NEWLINE	:	0515
		50	A7	9F	000F6		PUSHAB	P.AAG	:	0516
	68		01	FB	000F9		CALLS	#1, DBG\$PRINT	:	
	05		A2	E9	000FC	13\$:	BLBC	40(MODRSTPTR), 14\$:	0523
		58	A7	9F	00100		PUSHAB	P.AAH	:	0525
			03	11	00103		BRB	15\$:	
		64	A7	9F	00105	14\$:	PUSHAB	P.AAI	:	0528
	68		01	FB	00108	15\$:	CALLS	#1, DBG\$PRINT	:	
	53		A2	D0	0010B		MOVL	12(MODRSTPTR), DSTPTR	:	0536
	29	A2	03	A3	90	0010F	MOVB	3(DSTPTR), 41(MODRSTPTR)	:	0537
	20		56	E9	00114		BLBC	LANG CHANGE, 17\$:	0538
08	28	A2		04	E1	00117	BBC	#4, 40(MODRSTPTR), 16\$:	0541
			70	A7	9F	0011C	PUSHAB	P.AAJ	:	0543
	68		01	FB	0011F		CALLS	#1, DBG\$PRINT	:	
			13	11	00122		BRB	17\$:	
	7E		A2	9A	00124	16\$:	MOVZBL	41(MODRSTPTR), -(SP)	:	0546
	00000000G	00	01	FB	00128		CALLS	#1, DBG\$LANGUAGE	:	
			50	DD	0012F		PUSHL	R0	:	
			7C	A7	9F	00131	PUSHAB	P.AAK	:	0545
	68		02	FB	00134		CALLS	#2, DBG\$PRINT	:	
		20	A2	DD	00137	17\$:	PUSHL	32(MODRSTPTR)	:	0549
		0084	C7	9F	0013A		PUSHAB	P.AAL	:	
	68		02	FB	0013E		CALLS	#2, DBG\$PRINT	:	
	69		00	FB	00141		CALLS	#0, DBG\$NEWLINE	:	0550
	52		10	A2	D0	00144	MOVL	16(MODRSTPTR), MODRSTPTR	:	0551
			FF64	31	00148		BRW	9\$:	0492
	69		00	FB	0014B	18\$:	CALLS	#0, DBG\$NEWLINE	:	0560
		008C	C7	9F	0014E		PUSHAB	P.AAM	:	0561
	68		01	FB	00152		CALLS	#1, DBG\$PRINT	:	
	12		56	E8	00155		BLBS	LANG_CHANGE, 19\$:	0562
			55	DD	00158		PUSHL	LANG_CODE	:	0564
	00000000G	00	01	FB	0015A		CALLS	#1, DBG\$LANGUAGE	:	
			50	DD	00161		PUSHL	R0	:	
			0094	C7	9F	00163	PUSHAB	P.AAN	:	
	68		02	FB	00167		CALLS	#2, DBG\$PRINT	:	
			54	DD	0016A	19\$:	PUSHL	NUMBER_MODULES	:	0566
		009C	C7	9F	0016C		PUSHAB	P.AAO	:	
	68		02	FB	00170		CALLS	#2, DBG\$PRINT	:	
	00000000G	00	00	FB	00173		CALLS	#0, DBG\$FREE_MEM_LEFT	:	0567
7E	50		02	78	0017A		ASHL	#2, R0, -(SP)	:	
			00AC	C7	9F	0017E	PUSHAB	P.AAP	:	
	68		02	FB	00182		CALLS	#2, DBG\$PRINT	:	
	69		00	FB	00185		CALLS	#0, DBG\$NEWLINE	:	0568
			04	00188			RET		:	0570

; Routine Size: 393 bytes. Routine Base: DBG\$CODE + 00CF


```

444 0571 1 GLOBAL ROUTINE DBG$SHOW_TYPE(FLAG) : NOVALUE =
445 0572 1 +-
446 0573 1 Functional Description:
447 0574 1
448 0575 1     Print out the type specified in a SET TYPE[/OVERRIDE] statement,
449 0576 1     depending on the value of the input parameter.
450 0577 1
451 0578 1 Formal Parameters:
452 0579 1
453 0580 1     A flag to indicate whether the override type or the default type is to
454 0581 1     be printed
455 0582 1
456 0583 1 Implicit Inputs:
457 0584 1
458 0585 1     The type information stored in dbg$gl_dflttyp, dbg$gw_dfltleng,
459 0586 1     dbg$gbl_gbltyp, dbg$gw_gbllength.
460 0587 1
461 0588 1 Return Value:
462 0589 1
463 0590 1     NOVALUE
464 0591 1
465 0592 1 --
466 0593 1
467 0594 2 BEGIN
468 0595 2 LOCAL
469 0596 2     LENGTH,           ! local storage for the appropriate
470 0597 2     TYPE;             ! length and type values.
471 0598 2
472 0599 2
473 0600 2     ! Set up to use a new output buffer,
474 0601 2     ! and encode the standard beginning of
475 0602 2     ! the SHOW TYPE message into it.
476 0603 2
477 0604 2 dbg$flushbuf();
478 0605 2
479 0606 2     ! Insert the standard prefix.
480 0607 2
481 0608 2 CASE .flag FROM default TO override OF
482 0609 2     SET
483 0610 2     [default] :
484 0611 2         BEGIN
485 0612 2         dbg$print (UPLIT( %ASCIC 'type: '));
486 0613 2         type = .dbg$gl_dflttyp;
487 0614 2         length = .dbg$gw_dfltleng;
488 0615 2         END;
489 0616 2
490 0617 2     [override] :
491 0618 2         BEGIN
492 0619 2         dbg$print (UPLIT( %ASCIC 'type/override: '));
493 0620 2         type = .dbg$gl_gbltyp;
494 0621 2         length = .dbg$gw_gbllength;
495 0622 2         END;
496 0623 2     TES;
497 0624 2
498 0625 2     ! We print out a character string which indicates the type.
499 0626 2
500 0627 2 CASE .type FROM -1 TO dbg$k_maximum_dtype OF

```

```

501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553

```

```

0628
0629
0630
0631
0632
0633
0634
0635
0636
0637
0638
0639
0640
0641
0642
0643
0644
0645
0646
0647
0648
0649
0650
0651
0652
0653
0654
0655
0656
0657
0658
0659
0660
0661
0662
0663
0664
0665
0666
0667
0668
0669
0670
0671
0672
0673
0674
0675
0676
0677
0678
0679
0680

```

```

SET
[-1] :
      dbg$print (UPLIT( %ASCIC 'none' ));
[dsc$k_dtype_ac] :
      dbg$print (UPLIT( %ASCIC 'ascic' ));
[dsc$k_dtype_t, dsc$k_dtype_p] :
      BEGIN
          ! put keyword in buffer
          !
          IF .type EQL dsc$k_dtype_t
          THEN
              dbg$print(UPLIT( %ASCIC 'ascii: '))
          ELSE
              dbg$print(UPLIT( %ASCIC 'packed: '));
          !
          ! output the value in the current radix
          !
          DBG$OUT_NUM_VAL(.LENGTH);
      END;
[dsc$k_dtype_az] :
      dbg$print (UPLIT( %ASCIC 'asciz' ));
[dsc$k_dtype_b] :
      dbg$print (UPLIT( %ASCIC 'byte integer' ));
[dsc$k_dtype_d] :
      dbg$print (UPLIT( %ASCIC 'd_float' ));
[dsc$k_dtype_f] :
      dbg$print (UPLIT( %ASCIC 'f_float' ));
[dsc$k_dtype_g] :
      dbg$print (UPLIT( %ASCIC 'g_float' ));
[dsc$k_dtype_h] :
      dbg$print (UPLIT( %ASCIC 'h_float' ));
[dsc$k_dtype_l] :
      dbg$print (UPLIT( %ASCIC 'long integer' ));
[dsc$k_dtype_o] :
      dbg$print (UPLIT( %ASCIC 'octaword integer' ));
[dsc$k_dtype_q] :
      dbg$print (UPLIT( %ASCIC 'quadword integer' ));
[dsc$k_dtype_w] :
      dbg$print (UPLIT( %ASCIC 'word integer' ));
[dsc$k_dtype_zi] :
      dbg$print (UPLIT( %ASCIC 'instruction' ));
[dsc$k_dtype_adt] :
      dbg$print (UPLIT( %ASCIC 'date_time' ));
[inrange, outrange] :
      $DBG_ERROR('DBGSSV\DBG$SHOW_TYPE');
TES;
! output the completed buffer.
!
dbg$newline();
! of DBG$SHOW_TYPE.
END;

```

3A	65	64	69	72	72	65	00	20	3A	65	70	79	74	06	000CC	P.AAQ:	.ASCII	<6>\type: \<0>	
							76	6F	2F	65	70	79	74	0F	000D4	P.AAR:	.ASCII	<15>\type/override: \	
							00	00	00	65	6E	6F	6E	04	000E3				
							00	00	63	69	63	73	61	05	000E4	P.AAS:	.ASCII	<4>\none\<0><0><0>	
							20	3A	69	69	63	73	61	07	000EC	P.AAT:	.ASCII	<5>\ascic\<0><0>	
			00	00	00	20	3A	64	65	68	63	61	70	08	000F4	P.AAU:	.ASCII	<7>\ascii: \	
							00	00	7A	69	63	73	61	05	000FC	P.AAV:	.ASCII	<8>\packed: \<0><0><0>	
00	00	72	65	67	65	74	6E	69	20	65	74	79	62	0C	00108	P.AAW:	.ASCII	<5>\asciz\<0><0>	
														00	00110	P.AAX:	.ASCII	<12>\byte integer\<0><0><0>	
							74	61	6F	6C	66	5F	64	07	0011F				
							74	61	6F	6C	66	5F	66	07	00120	P.AAY:	.ASCII	<7>\d_float\	
							74	61	6F	6C	66	5F	67	07	00128	P.AAZ:	.ASCII	<7>\f_float\	
							74	61	6F	6C	66	5F	68	07	00130	P.ABA:	.ASCII	<7>\g_float\	
00	00	72	65	67	65	74	6E	69	20	67	6E	6F	6C	0C	00138	P.ABB:	.ASCII	<7>\h_float\	
														00	00140	P.ABC:	.ASCII	<12>\long integer\<0><0><0>	
														00	0014F				
67	65	74	6E	69	20	64	72	6F	77	61	74	63	6F	10	00150	P.ABD:	.ASCII	<16>\octaword integer\<0><0><0>	
										00	00	00	72	65	0015F				
67	65	74	6E	69	20	64	72	6F	77	64	61	75	71	10	00164	P.ABE:	.ASCII	<16>\quadword integer\<0><0><0>	
										00	00	00	72	65	00173				
00	00	72	65	67	65	74	6E	69	20	64	72	6F	77	0C	00178	P.ABF:	.ASCII	<12>\word integer\<0><0><0>	
														00	00187				
			6E	6F	69	74	63	75	72	74	73	6E	69	08	00188	P.ABG:	.ASCII	<11>\instruction\	
			00	00	65	6D	69	74	5F	65	74	61	64	09	00194	P.ABH:	.ASCII	<9>\date time\<0><0>	
4F	48	53	24	47	42	44	5C	56	53	53	47	42	44	14	001A0	P.ABI:	.ASCII	<20>\DBGSSV\<92>\DBG\$SHOW_TYPE\	
									45	50	59	54	5F	57	001AF				

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

										001C	00000					.ENTRY	DBG\$SHOW_TYPE, Save R2,R3,R4		0571
									54	00000000G	00	9E	00002			MOVAB	DBG\$PRINT, R4		
									53	00000000'	EF	9E	00009			MOVAB	P.AAQ, R3		
							01	00000000G	00		00	FB	00010			CALLS	#0, DBG\$FLUSHBUF		0604
									00		04	AC	CF	00017		CASEL	FLAG, #0, #1		0608
									0019		0004		0001C	1\$:		.WORD	2\$-1\$,-		
																	3\$-1\$		
											53	DD	00020	2\$:		PUSHL	R3		0612
									64		01	FB	00022			CALLS	#1, DBG\$PRINT		
									50	00000000G	00	DD	00025			MOVL	DBG\$GL_DFLTYP, TYPE		0613
									52	00000000G	00	3C	0002C			MOVZWL	DBG\$GW_DFLTLENG, LENGTH		0614
											14	11	00033			BRB	4\$		0608
										08	A3	9F	00035	3\$:		PUSHAB	P.AAR		0619
									64		01	FB	00038			CALLS	#1, DBG\$PRINT		
									50	00000000G	00	DD	0003B			MOVL	DBG\$GL_GBLTYP, TYPE		0620
									52	00000000G	00	3C	00042			MOVZWL	DBG\$GW_GBLLENG, LENGTH		0621
							2C	FFFFFFFF	8F		50	CF	00049	4\$:		CASEL	TYPE, #-1, #44		0627
														5\$:		.WORD	7\$-5\$,-		
005A							005A				006F		00051				6\$-5\$,-		
0097							005A				005A		00059				6\$-5\$,-		
00A1							00BB				00C1		00061				6\$-5\$,-		
0079							005A				009C		00069				6\$-5\$,-		
005A							005A				005A		00071				6\$-5\$,-		
00C7							0079				005A		00079				6\$-5\$,-		
00B5							005A				005A		00081				6\$-5\$,-		
005A							005A				00AB		00089				13\$-5\$,-		

005A
0074
005A

005A
005A
005A

005A
005A
005A

005A 00091
00CD 00099
0092 000A1
005A 000A9

21\$-5\$,-
18\$-5\$,-
20\$-5\$,-
15\$-5\$,-
14\$-5\$,-
6\$-5\$,-
6\$-5\$,-
9\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
9\$-5\$,-
22\$-5\$,-
6\$-5\$,-
6\$-5\$,-
19\$-5\$,-
16\$-5\$,-
17\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
23\$-5\$,-
6\$-5\$,-
6\$-5\$,-
8\$-5\$,-
12\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$,-
6\$-5\$

		00D4	C3	9F	000AB	6\$:	PUSHAB	P.ABI	0674
			01	DD	000AF		PUSHL	#1	
		00028362	8F	DD	000B1		PUSHL	#164706	
00000000G	00		03	FB	000B7		CALLS	#3, LIB\$SIGNAL	
			65	11	000BE		BRB	25\$	
		18	A3	9F	000C0	7\$:	PUSHAB	P.AAS	0630
			5D	11	000C3		BRB	24\$	
		20	A3	9F	000C5	8\$:	PUSHAB	P.AAT	0632
			58	11	000C8		BRB	24\$	
		OE	50	D1	000CA	9\$:	C MPL	TYPE, #14	0638
			05	12	000CD		BNEQ	10\$	
		28	A3	9F	000CF		PUSHAB	P.AAU	0640
			03	11	000D2		BRB	11\$	
		30	A3	9F	000D4	10\$:	PUSHAB	P.AAV	0642
		64	01	FB	000D7	11\$:	CALLS	#1, DBG\$PRINT	
			52	DD	000DA		PUSHL	LENGTH	0646
FCC7	CF		01	FB	000DC		CALLS	#1, DBG\$OUT_NUM_VAL	
			42	11	000E1		BRB	25\$	0627
		3C	A3	9F	000E3	12\$:	PUSHAB	P.AAW	0650
			3A	11	000E6		BRB	24\$	

	44	A3	9F	000E8	13\$:	PUSHAB	P.AAX		: 0652
		35	11	000EB		BRB	24\$:
	54	A3	9F	000ED	14\$:	PUSHAB	P.AAY		: 0654
		30	11	000F0		BRB	24\$:
	5C	A3	9F	000F2	15\$:	PUSHAB	P.AAZ		: 0656
		28	11	000F5		BRB	24\$:
	64	A3	9F	000F7	16\$:	PUSHAB	P.ABA		: 0658
		26	11	000FA		BRB	24\$:
	6C	A3	9F	000FC	17\$:	PUSHAB	P.ABB		: 0660
		21	11	000FF		BRB	24\$:
	74	A3	9F	00101	18\$:	PUSHAB	P.ABC		: 0662
		1C	11	00104		BRB	24\$:
	0084	C3	9F	00106	19\$:	PUSHAB	P.ABD		: 0664
		16	11	0010A		BRB	24\$:
	0098	C3	9F	0010C	20\$:	PUSHAB	P.ABE		: 0666
		10	11	00110		BRB	24\$:
	00AC	C3	9F	00112	21\$:	PUSHAB	P.ABF		: 0668
		0A	11	00116		BRB	24\$:
	00BC	C3	9F	00118	22\$:	PUSHAB	P.ABG		: 0670
		04	11	0011C		BRB	24\$:
	00C8	C3	9F	0011E	23\$:	PUSHAB	P.ABH		: 0672
		01	FB	00122	24\$:	CALLS	#1, DBG\$PRINT		:
	00000000G	00	FB	00125	25\$:	CALLS	#0, DBG\$NEWLINE		: 0679
		04		0012C		RET			: 0680

: Routine Size: 301 bytes, Routine Base: DBG\$CODE + 0258

: 554 0681 1 END
: 555 0682 0 ELUDOM

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes
DBG\$OWN	4	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
DBG\$PLIT	437	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)
DBG\$CODE	901	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
-\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	14	0	1000	00:01.9
-\$255\$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1	32	0	0	7	00:00.2
-\$255\$DUA28:[DEBUG.OBJ]DBGLIB.L32;1	1545	61	3	97	00:01.9
-\$255\$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1	418	107	25	31	00:00.3

DBGSSV
V04-000

1 12
16-Sep-1984 02:38:25
14-Sep-1984 12:17:46

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[DEBUG.SRC]DBGSSV.B32;1 Page 20 (7)

:	_\$255\$DUA28:[DEBUG.OBJ]DBGMSG.L32;1	386	5	1	22	00:00.3
:	_\$255\$DUA28:[DEBUG.OBJ]DBGGEN.L32;1	150	2	1	12	00:00.3

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

COMMAND QUALIFIERS

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

: Size: 901 code + 441 data bytes
: Run Time: 00:19.2
: Elapsed Time: 00:22.5
: Lines/CPU Min: 2127
: Lexemes/CPU-Min: 9160
: Memory Used: 186 pages
: Compilation Complete

