F 1

```
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEEEEEEEEEEE     BBBBBBBBBBBB     UUU         UUU  GGG
DDD         DDD  EEEEEEEEEEEE     BBBBBBBBBBBB     UUU         UUU  GGG
DDD         DDD  EEEEEEEEEEEE     BBBBBBBBBBBB     UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG   GGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG   GGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG   GGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG         GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG         GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG         GGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUUUUUUUUUUUUUU     GGGGGGGGG
```

```
                                    C 13
**FILE**ID**DBGLANVEC




DDDDDDDD    BBBBBBBB      GGGGGGGG  LL            AAAAAA    NN      NN  VV      VV  EEEEEEEEEE    CCCCCCCC
DDDDDDDD    BBBBBBBB      GGGGGGGG  LL            AAAAAA    NN      NN  VV      VV  EEEEEEEEEE    CCCCCCCC
DD      DD  BB      BB  GG          LL          AA      AA  NN      NN  VV      VV  EE          CC
DD      DD  BB      BB  GG          LL          AA      AA  NN      NN  VV      VV  EE          CC
DD      DD  BB      BB  GG          LL          AA      AA  NNNN    NN  VV      VV  EE          CC
DD      DD  BB      BB  GG          LL          AA      AA  NNNN    NN  VV      VV  EE          CC
DD      DD  BBBBBBBB    GG          LL          AA      AA  NN  NN  NN  VV      VV  EEEEEEEE    CC
DD      DD  BBBBBBBB    GG          LL          AA      AA  NN  NN  NN  VV      VV  EEEEEEEE    CC
DD      DD  BB      BB  GG  GGGGGG  LL          AAAAAAAAAA  NN    NNNN  VV      VV  EE          CC
DD      DD  BB      BB  GG  GGGGGG  LL          AAAAAAAAAA  NN    NNNN  VV      VV  EE          CC
DD      DD  BB      BB  GG      GG  LL          AA      AA  NN      NN    VV  VV    EE          CC              ....
DD      DD  BB      BB  GG      GG  LL          AA      AA  NN      NN    VV  VV    EE          CC              ....
DDDDDDDD    BBBBBBBB      GGGGGG    LLLLLLLLLL  AA      AA  NN      NN      VV      EEEEEEEEEE    CCCCCCCC      ....
DDDDDDDD    BBBBBBBB      GGGGGG    LLLLLLLLLL  AA      AA  NN      NN      VV      EEEEEEEEEE    CCCCCCCC      ....

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
```

```
0 MODULE DBGLANVEC (IDENT = 'V04-000') =
0
1 BEGIN
1 !
1 !****************************************************************************
1 !*                                                                          *
1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
1 !*  ALL RIGHTS RESERVED.                                                    *
1 !*                                                                          *
1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
1 !*  TRANSFERRED.                                                            *
1 !*                                                                          *
1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
1 !*  CORPORATION.                                                            *
1 !*                                                                          *
1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
1 !*                                                                          *
1 !*                                                                          *
1 !****************************************************************************
1
1 !
1 ! WRITTEN BY
1 !       Bruce Olsen     July, 1980
1 !
1 ! REWRITTEN BY
1 !       Rich Title      July, 1983
1 !
1 ! MODULE FUNCTION
1 !       This module contains several miscellaneous routines for
1 !       manipulating descriptors. The name of the module is a holdover
1 !       from the days when each language had its own Primary and Value
1 !       Descriptors. At that time, this module had routines which
1 !       did a CASE on the language, and called the appropriate language
1 !       routine. Now that we have common Primary and Value descriptors
1 !       for all languages, this is no longer necessary. But the routines
1 !       for copying descriptors, deleting descriptors, and so on,
1 !       still reside in this module.
1 !
1
1 ! MODIFIED BY
1 !       R. Title        Aug 1982        Put in code to check for implementationm
1 !                                       level = 3, so that we can test new support
1 !                                       for PASCAL, PLI, and COBOL.
1 !       R. Title        Aug 1982        Added comments to each routine so that
1 !                                       the description now says what the routine
1 !                                       does, instead of just saying "see the
1 !                                       language-specific routines".
1 !       R. Title        Mar 1983        Removed all of the "level 2" PASCAL,
1 !                                       PL/I and COBOL code.
1 !
```

```
;    58          0058  1 REQUIRE 'SRC$:DBGPROLOG.REQ';
;    59          0192  1
;    60          0193  1 FORWARD ROUTINE
;    61          0194  1     DBG$NGET_LVAL,
;    62          0195  1     DBG$NGET_TYPE,
;    63          0196  1     DBG$NMAKE_VAL_DESC,
;    64          0197  1     DBG$NTYPE_CONV,
;    65          0198  1     DBG$NSYMBOLIZE,
;    66          0199  1     DBG$NGET_PAGES,
;    67          0200  1     DBG$NGET_LENGTH,
;    68          0201  1     DBG$NCOPY_DESC,
;    69          0202  1     COPY_DESC_HANDLER,
;    70          0203  1     DBG$NFREE_DESC,
;    71          0204  1     DBG$NGET_SYMID,
;    72          0205  1     DBG$NINITIALIZE: NOVALUE;
;    74      0206  1 EXTERNAL ROUTINE
;    75      0207  1     DBG$DATA_LENGTH,                       ! Obtain length from VMS descriptor
;    76      0208  1     DBG$EVAL_LANG_OPERATOR,                ! Evaluate operator expressions in
;    77      0209  1                                            !       current language
;    78      0210  1     DBG$GET_MEMORY,                        ! Allocate permanent memory
;    79      0211  1     DBG$GET_TEMPMEM,                       ! Allocate temporary memory
;    80      0212  1     DBG$MAKE_VMS_DESC,                     ! Convert Primary Descriptor to
;    81      0213  1                                            !       VMS descriptor
;    82      0214  1     DBG$PRIM_TO_VAL,                       ! Convert Primary Descriptor to
;    83      0215  1                                            !       Value Descriptor
;    84      0216  1     DBG$PRINT_AGGREGATE: NOVALUE,          ! Output an aggregate object
;    85      0217  1     DBG$PRINT_IDENTIFIER,                  ! Replacement for DBG$NSYMBOLIZE -
;    86      0218  1                                            !       prints an identifier.
;    87      0219  1     DBG$PRINT_VALUE: NOVALUE,              ! Print a value descriptor.
;    88      0220  1     DBG$REL_MEMORY: NOVALUE;               ! Release memory
;    89      0221  1
;    90      0222  1 EXTERNAL
;    91      0223  1     DBG$GB_LANGUAGE : BYTE,                ! Language code for current language
;    92      0224  1     DBG$GL_CONVERT_TOKEN,                  ! Pointer to CONVERT token
;    93      0225  1     DBG$GL_DEPOSIT_TOKEN;                  ! Pointer to DEPOSIT token
;    94      0226  1
;    95      0227  1 LITERAL
;    96      0228  1     MIN_LANGUAGE_CODE = MIN (DBG$K_PLI, DBG$K_PASCAL, DBG$K_COBOL),
;    97      0229  1     MAX_LANGUAGE_CODE = MAX (DBG$K_PLI, DBG$K_PASCAL, DBG$K_COBOL);
;    98      0230  1
;    99      0231  1 OWN
;   100      0232  1         COPY_DESC_HEAD;                ! Points to the header of a copied descriptor,
;   101      0233  1                                        ! if we are copying the descriptor into
;   102      0234  1                                        ! permanent memory. This is used by
;   103      0235  1                                        ! COPY_DESC_HANDLER.
;   104      0236  1
```

```
1 GLOBAL ROUTINE DBG$NGET_LVAL (PRIM_DESC, PARAM2, PARAM3) =
1 !
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       Obtains a symbol's lvalue using the primary descriptor for that
1 !       symbol. Note that most types of named constants do not have an
1 !       lvalue. The debugger gives special treatment to named constants
1 !       which have read only memory allocated to contain their value.
1 !
1 !       This routine is still called from DBGEXC,
1 !       in the process of displaying "old value", "new value" on watchpoints.
1 !       This routine can thus go away when DBGEXC is replaced by DBGEVENT.
1 !
1 ! FORMAL PARAMETERS:
1 !
1 !       prim_desc         - A longword which contains the address of a primary descriptor
1 !
1 !       param2            - The address of a quadword to contain the lvalue of the
1 !                           entity described by the primary descriptor and the bit
1 !                           offset, if any. The byte address will be contained in
1 !                           in the first longword, the bit offset in the second
1 !                           longword.
1 !
1 !       param3            - The address of a longword to contain the address of
1 !                           a message argument vector as described on page 4-119
1 !                           of the VAX/VMS system reference, volume 1A
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       NONE
1 !
1 ! IMPLICIT OUTPUTS:
1 !
1 !       NONE
1 !
1 ! ROUTINE VALUE:
1 !
1 !       An unsigned longword integer completion code
1 !
1 ! COMPLETION CODES:
1 !
1 !       STS$K_SUCCESS (1) - Success. The object described by the input primary
1 !                           descriptor has an lvalue which is being returned.
1 !
1 !       STS$K_ERROR   (2) - Failure. Object does not have an lvalue.
1 !
1 ! SIDE EFFECTS:
1 !
1 !       NONE
1 !
1 !
2   BEGIN
2
2   MAP
2       PRIM_DESC : REF DBG$PRIMARY;    ! Points to a new style Primary
2                                       ! Descriptor.
```

```
;      2      LOCAL
;      2          VMS_DESC: REF DBG$STG_DESC,
;      2          VMS_DESC_AREA: DBG$STG_DESC;
;      2
;      2
;      2      IF .PRIM_DESC [DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC
;      2      THEN
;      3          BEGIN
;      3
;      3          ! Set up the VMS descriptor.
;      3          !
;      3          VMS_DESC = VMS_DESC_AREA;
;      3
;      3          ! Call the routine that fills in the VMS descriptor.
;      3          !
;      3          DBG$MAKE_VMS_DESC (.PRIM_DESC, .VMS_DESC);
;      3          END
;      3
;      3      ! Value descriptor or volatile value descriptor - we already
;      3      ! have a VMS descriptor.
;      3      !
;      2      ELSE IF .PRIM_DESC [DBG$B_DHDR_TYPE] EQL DBG$K_VALUE_DESC
;      2      OR .PRIM_DESC [DBG$B_DHDR_TYPE] EQL DBG$K_V_VALUE_DESC
;      2      THEN
;      2          VMS_DESC = PRIM_DESC [DBG$A_VALUE_VMSDESC]
;      2
;      2      ! We do not expect any other kind of descriptor.
;      2      !
;      2      ELSE
;      2          $DBG_ERROR ('DBGLANVEC\DBG$NGET_LVAL unknown descriptor kind');
;      2
;      2      ! Fill in the output parameter to point to the
;      2      ! (byte address, bit offset) quadword in the VMS descriptor.
;      2      !
;      2      .PARAM2     = .VMS_DESC[DSC$A_POINTER];
;      2      .PARAM2 + 4 = .VMS_DESC[DSC$L_POS];
;      2      RETURN STS$K_SUCCESS;
;      1      END;


                                                                       .TITLE  DBGLANVEC
                                                                       .IDENT  \V04-000\

                                                                       .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

24 47 42 44 5C 43 45 56 4E 41 4C 47 42 44 2F 00000 P.AAA:  .ASCII  \/DBGLANVEC\<92>\DBG$NGET_LVAL unknown d\    ;
6F 6E 6B 6E 75 20 4C 41 56 4C 5F 54 45 47 4E 0000F                                                            ;
                                    64 20 6E 77 0001E                                                            ;
   64 6E 69 6B 20 72 6F 74 70 69 72 63 73 65 00022          .ASCII  \escriptor kind\                          ;

                                                                       .PSECT  DBG$OWN,NOEXE,  PIC,2

                                                     00000 COPY_DESC_HEAD:
                                                                       .BLKB   4

                                                                       .EXTRN  DBG$DATA_LENGTH
                                                                       .EXTRN  DBG$EVAL_LANG_OPERATOR
```

```
                                                            .EXTRN  DBG$GET_MEMORY, DBG$GET_TEMPMEM
                                                            .EXTRN  DBG$MAKE_VMS_DESC
                                                            .EXTRN  DBG$PRIM_TO_VAL
                                                            .EXTRN  DBG$PRINT_AGGREGATE
                                                            .EXTRN  DBG$PRINT_IDENTIFIER
                                                            .EXTRN  DBG$PRINT_VALUE
                                                            .EXTRN  DBG$REL_MEMORY, DBG$GB_LANGUAGE
                                                            .EXTRN  DBG$GL_CONVERT_TOKEN
                                                            .EXTRN  DBG$GL_DEPOSIT_TOKEN

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

                                             0004 00000     .ENTRY  DBG$NGET_LVAL, Save R2                          ; 0237
                                   5E     0C   C2 00002     SUBL2   #12, SP                                         ;
00000079  8F      04  BC           08     10   ED 00005     CMPZV   #16, #8, @PRIM_DESC, #121                       ; 0299
                                          11   12 0000F     BNEQ    1$                                              ;
                                   52     6E   9E 00011     MOVAB   VMS_DESC_AREA, VMS_DESC                         ; 0305
                                          52   DD 00014     PUSHL   VMS_DESC                                        ; 0309
                                04        AC   DD 00016     PUSHL   PRIM_DESC                                       ;
                      00000000G    00     02   FB 00019     CALLS   #2, DBG$MAKE_VMS_DESC                           ;
                                          34   11 00020     BRB     4$                                              ; 0299
0000007A  8F      04  BC           08     10   ED 00022 1$: CMPZV   #16, #8, @PRIM_DESC, #122                       ; 0315
                                          0C   13 0002C     BEQL    2$                                              ;
00000083  8F      04  BC           08     10   ED 0002E     CMPZV   #16, #8, @PRIM_DESC, #131                       ; 0316
                                          07   12 00038     BNEQ    3$                                              ;
                      52       04  AC     14   C1 0003A 2$: ADDL3   #20, PRIM_DESC, VMS_DESC                        ; 0318
                                          15   11 0003F     BRB     4$                                              ;
                                00000000' EF   9F 00041 3$: PUSHAB  P.AAA                                           ; 0323
                                          01   DD 00047     PUSHL   #1                                              ;
                                00028362  8F   DD 00049     PUSHL   #164706                                         ;
                      00000000G    00     03   FB 0004F     CALLS   #3, LIB$SIGNAL                                  ;
                                   50  08 AC   D0 00056 4$: MOVL    PARAM2, R0                                      ; 0328
                                   60  04 A2   7D 0005A     MOVQ    4(VMS_DESC), (R0)                               ;
                                   50     01   D0 0005E     MOVL    #1, R0                                          ; 0330
                                               04 00061     RET                                                     ; 0331

; Routine Size:  98 bytes,    Routine Base:  DBG$CODE + 0000
```

```
GLOBAL ROUTINE DBG$NGET_TYPE (PRIM_DESC, PARAM2, PARAM3) =
!
! FUNCTIONAL DESCRIPTION:
!
!       Uses a symbol's primary descriptor to return type information. The
!       types recognized are limited to three:
!
!       1)      - type named constant and instruction
!                 (lexical entities, labels)
!
!       2)      - type named constant and
!                 noinstruction (symbolic literals)
!
!       3)      - type other
!
!       This routine is still called from DBGEXC.
!       It can go away when we convert over to the new DBGEVENT.
!
! FORMAL PARAMETERS:
!
!       prim_desc       - A longword containing the address of a primary descriptor
!
!       param2          - The address of a longword to contain an unsigned integer
!                         encoding of the symbol's type as follows:
!
!               dbg$k_nc_instruction (125)      - named constant, instruction
!
!               dbg$k_nc_other (126)            - named constant, noinstruction
!
!               dbg$k_other (127)               - other
!
!       param3          - The address of a longword to contain the address of
!                         a message argument vector as described on page 4-119
!                         of the VAX/VMS system reference, volume 1A
!
! IMPLICIT INPUTS:
!
!       NONE
!
! IMPLICIT OUTPUTS:
!
!       In case of a severe error return, a message argument vector is constructed
!       from dynamic storage and returned.
!
! ROUTINE VALUE:
!
!       An unsigned integer longword completion code
!
! COMPLETION CODES:
!
!       STS$K_SUCCESS (1) - Success. Type information recovered and returned.
!
!       STS$K_SEVERE  (4) - Failure. No type information recovered. Message
!                           argument vector constructed and returned.
!
! SIDE EFFECTS:
!
```

```
;  259          0389 1 !       NONE
;  260          0390 1 !
;  261          0391 2     BEGIN
;  262          0392 2
;  263          0393 2     ! For now, always return "OTHER". This may not be completely
;  264          0394 2     ! correct - we will fix it up later.
;  265          0395 2     !
;  266          0396 2     .PARAM2 = DBG$K_OTHER;
;  267          0397 2     RETURN STS$K_SUCCESS;
;  268          0398 1     END;



                                  0000 00000          .ENTRY  DBG$NGET_TYPE, Save nothing                     ; 0332
                 08   BC     7F   8F  9A 00002          MOVZBL  #127, @PARAM2                                  ; 0396
                      50          01  D0 00007          MOVL    #1, R0                                         ; 0397
                                      04 0000A          RET                                                    ; 0398

; Routine Size:  11 bytes,    Routine Base:  DBG$CODE + 0062
```

```
1 GLOBAL ROUTINE DBG$NMAKE_VAL_DESC (PRIM_DESC, PARAM2, PARAM3, PARAM4) =
1 !
1 !++
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       Translates language specific primary descriptors to language specific
1 !       value descriptors. This routine should be able to use the symbol table
1 !       access routines and the information contained within the primary descriptor
1 !       to construct a descriptor which represents a 'value materialization' for
1 !       the object represented by the input primary descriptor.
1 !
1 !       Note that this routine must be able to use life-time, invocation, and
1 !       generation information to produce an accurate value descriptor of the
1 !       input object, or to decide when the value of an object cannot be
1 !       materialized (such as when the user's PC is not within the scope of
1 !       a dynamic variable).
1 !
1 !       Value descriptors produced by this routine must be marked (within the
1 !       type field of the language independent header block) as to whether
1 !       they are non-volatile (dsc$k_value_desc) or volatile (dsc$k_v_value_desc).
1 !       Volatile value descriptors will NOT be stored to represent '\', 'last value'.
1 !
1 !       Since value descriptors may be used as target descriptors ( as input to
1 !       dbg$npli_type_conv ), some provision must be made for incorporating
1 !       a value pointer field within the value descriptor. This type of value
1 !       descriptor is loosely defined as a volatile type.
1 !
1 !       This routine is still called from DBGEXC in the process of giving
1 !       watchpoint display. It can thus go away when DBGEXC is replaced
1 !       by DBGEVENT.
1 !
1 !       This routine call a language-specific routine based on the language
1 !       code in the descriptor header.
1 !
1 ! FORMAL PARAMETERS:
1 !
1 !       prim_desc       - A longword containing the address of a primary descriptor
1 !
1 !       param2          - A longword containing boolean true or false. When true,
1 !                         the caller is requesting the construction of a value
1 !                         descriptor that can be used as a target descriptor for
1 !                         the type converter. The resulting value must therefore
1 !                         contain a pointer to the value of the entity described
1 !                         by the input primary descriptor. Presumably, such a
1 !                         value descriptor will be of volatile type.
1 !
1 !       param3          - The address of a longword to contain the address of the
1 !                         resulting value descriptor
1 !
1 !       param4          - The address of a longword to contain the address of a
1 !                         message argument vector as described on page 4-119 of
1 !                         the VAX/VMS system reference, volume 1A
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       Depends on the language-specific routine.
1 !
```

```
1 ! IMPLICIT OUTPUTS:
1 !
1 !       In case of a success return, the resulting value descriptor must be
1 !       constructed from dynamic storage and returned.
1 !
1 !       In case of a severe error return, a message argument vector must be
1 !       constructed from dynamic storage and returned.
1 !
1 ! ROUTINE VALUE:
1 !
1 !       An unsigned integer longword completion code
1 !
1 ! COMPLETION CODES:
1 !
1 !       STS$K_SUCCESS (1) - Success. Value descriptor constructed and returned.
1 !
1 !       STS$K_SEVERE  (4) - Failure. Value descriptor not constructed. Message
1 !                           argument vector constructed and returned.
1 !
1 ! SIDE EFFECTS:
1 !
1 !       NONE
1 !
2     BEGIN
2
2     MAP
2         PRIM_DESC: REF DBG$PRIMARY;
2
2     ! Don't convert to value desc if the primary is an aggregate.
2     !
2     IF .PRIM_DESC [DBG$V_DHDR_AGGR]
2     THEN
2         .PARAM3 = .PRIM_DESC
2     ELSE
2         IF NOT DBG$PRIM_TO_VAL (
2                 .PRIM_DESC,
2                 (IF .PARAM2 THEN DBG$K_V_VALUE_DESC ELSE DBG$K_VALUE_DESC),
2                 .PARAM3)
2         THEN
2             $DBG_ERROR ('DBGLANVEC\DBG$NMAKE_VAL_DESC bad return code from PRIM_TO_VAL');
2     RETURN STS$K_SUCCESS;
1     END;


                                                                           .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

24 47 42 44 5C 43 45 56 4E 41 4C 47 42 44 3D 00030 P.AAB:  .ASCII  \=DBGLANVEC\<92>\DBG$NMAKE_VAL_DESC bad \      ;
20 43 53 45 44 5F 4C 41 56 5F 45 4B 41 4D 4E 0003F                                                             ;
                                    20 64 61 62 0004E                                                             ;
6F 72 66 20 65 64 6F 63 20 6E 72 75 74 65 72 00052          .ASCII  \return code from PRIM_TO_VAL\            ;
      4C 41 56 5F 4F 54 5F 4D 49 52 50 20 6D 00061                                                             ;


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

```
                                        0000 00000          .ENTRY  DBG$NMAKE_VAL_DESC, Save nothing          : 0399
                   50          04   AC   D0 00002          MOVL    PRIM_DESC, R0                              : 0486
                   06          04   A0   E9 00006          BLBC    4(R0), 1$                                  :
            0C     BC               50   D0 0000A          MOVL    R0, @PARAM3                                : 0488
                                    32   11 0000E          BRB     4$                                         :
                               0C   AC   DD 00010 1$:      PUSHL   PARAM3                                     : 0493
                   06          08   AC   E9 00013          BLBC    PARAM2, 2$                                 : 0492
                   7E          83   8F   9A 00017          MOVZBL  #131, -(SP)                                :
                                    04   11 0001B          BRB     3$                                         :
                   7E          7A   8F   9A 0001D 2$:      MOVZBL  #122, -(SP)                                :
                                    50   DD 00021 3$:      PUSHL   R0                                         : 0491
     00000000G     00               03   FB 00023          CALLS   #3, DBG$PRIM_TO_VAL                        :
                   15               50   E8 0002A          BLBS    R0, 4$                                     :
                         00000000'  EF   9F 0002D          PUSHAB  P.AAB                                      : 0495
                                    01   DD 00033          PUSHL   #1                                         :
                         00028362   8F   DD 00035          PUSHL   #164706                                    :
     00000000G     00               03   FB 0003B          CALLS   #3, LIB$SIGNAL                             :
                   50               01   D0 00042 4$:      MOVL    #1, R0                                     : 0496
                                         04 00045          RET                                                : 0497

; Routine Size:  70 bytes,    Routine Base:  DBG$CODE + 006D
```

```
GLOBAL ROUTINE DBG$NTYPE_CONV (VALUE_DESC, PARAM2, PARAM3, PARAM4, PARAM5) =
!
! FUNCTIONAL DESCRIPTION:
!
!       Performs language specific and language independent type conversions.
!       These will be both internal-to-internal and internal-to-external in
!       nature. Target may be described by either language
!       specific value descriptor or a subset of VAX standard descriptors.
!       The latter category includes the following:
!
!       dsc$k_dtype_v
!
!       dsc$k_dtype_b, dsc$k_dtype_bu
!
!       dsc$k_dtype_w, dsc$k_dtype_wu
!
!       dsc$k_dtype_l, dsc$k_dtype_lu
!
!       dsc$k_dtype_q, dsc$k_dtype_qu
!
!       dsc$k_dtype_f, dsc$k_dtype_d
!
!       dsc$k_dtype_t
!
!       The source descriptor must be a language specific value descriptor.
!
!       Note that this routine will be used to obtain the 'printable' (external)
!       value of the source as the result of EXAMINE commands.
!
!       This routine is still called from a couple of places; one is to
!       convert the expression in an IF or a WHILE command to boolean;
!       another is to display the value of watchpoints in "old value",
!       "new value" displays. (This second use of this routine will go
!       away when DBGEVENT replaces DBGEXC.)
!
! FORMAL PARAMETERS:
!
!       value_desc      - A longword which contains the address of
!                         a language specific value descriptor
!
!       param2          - A longword containing an integer encoding of the radix
!                         to be used when converting to a 'printable' value:
!
!               dbg$k_default (1)       - source language default radix
!
!               dbg$k_binary (2)        - binary radix
!
!               dbg$k_octal (8)         - octal radix
!
!               dbg$k_decimal (10)      - decimal radix
!
!               dbg$k_hex (16)          - hexadecimal radix
!
!                       Note that this parameter is significant ONLY when the
!                       object described by the source descriptor is to be
!                       converted to external format. A request for a binary,
!                       octal, or hex 'printable' value means to consider the
```

```
1 !                         value of source as a bit pattern to be translated to
1 !                         special characters. In this sense, the type of the source
1 !                         value is not significant - only the length. Values will
1 !                         therefore be displayed as unsigned integers within the
1 !                         specified radix. Values will be left-extended to nibble
1 !                         boundaries.
1 !
1 !       param3            - A longword containing an unsigned integer encoding of the
1 !                           type of information contained within the target parameter:
1 !
1 !               dbg$k_vax_desc (130)        - target contains the address of a
1 !                                             VAX standard descriptor
1 !
1 !                                             Note: The caller of dbg$nxxx_type_conv
1 !                                             must assure that the dsc$a_pointer
1 !                                             field of the descriptor contains the
1 !                                             address of an appropriately large
1 !                                             block of storage.
1 !
1 !               dbg$k_value_desc (122)      - target contains the address of a
1 !                                             language specific value descriptor.
1 !                                             The type convertor deposits the
1 !                                             value of Source into the address of
1 !                                             the value in Target.
1 !
1 !               dbg$k_external_desc (129)   - target contains the address of
1 !                                             a VAX standard string descriptor.
1 !                                             This is a request to convert to
1 !                                             'printable' format. Conversion must
1 !                                             include check for unprintable characters.
1 !
1 !       param4            - A longword which contains the address of either a VAX
1 !                           standard descriptor, or a language specific value descriptor
1 !
1 !       param5            - The address of a longword to contain the address of
1 !                           a message argument vector as described on page 4-119 of
1 !                           the VAX/VMS system reference, volume 1A
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       NONE
1 !
1 ! IMPLICIT OUTPUTS:
1 !
1 !       When this routine is called to obtain the 'printable' (external) value
1 !       of the source object, the target will contain the address of a VAX
1 !       standard string descriptor with length and pointer fields set to 0.
1 !       Dynamic storage must be obtained to contain the resulting ascii string.
1 !
1 !       In all other cases, this routine is not required to allocate storage to
1 !       contain the resulting value of a conversion request. Targets which are
1 !       described by VAX standard descriptors MUST contain the address of a
1 !       block of storage (the dsc$a_pointer field)
1 !       in which the resulting value of the conversion will be stored.
1 !
1 !       Dynamic storage must be used to construct the message argument vector
1 !       upon a severe error return.
```

```
1 !
1 ! ROUTINE VALUE:
1 !
1 !       unsigned integer longword completion code
1 !
1 ! COMPLETION CODES:
1 !
1 !       STS$K_SUCCESS (1) - Success. Conversion performed.
1 !
1 !       STS$K_SEVERE  (4) - Failure. No conversion. Message argument vector
1 !                           constructed and returned.
1 !
1 ! SIDE EFFECTS:
1 !
1 !       Informational messages such as string and number truncation may be
1 !       issued during processing.
1 !
1
2     BEGIN
2
2     SELECTONE .PARAM3 OF
2         SET
2
2         ! One place this routine is called is in the processing of the
2         ! IF, WHILE, and INCR commands, in order to convert the given
2         ! value to a type understood by the command.
2         ! In these cases, the third parameter is DBG$K_VAX_DESC and
2         ! the fourth parameter is a pointer to a VAX standard descriptor.
2         !
2         [DBG$K_VAX_DESC] :
3             BEGIN
3             LOCAL
3                 V_VAL_DESC: REF DBG$VALDESC;
3
3             ! Build a volatile value descriptor around the given VAX
3             ! standard descriptor.
3             !
3             V_VAL_DESC = DBG$GET_TEMPMEM (DBG$K_VALDESC_BASE_SIZE+4);
3             CH$MOVE (12, .VALUE_DESC, .V_VAL_DESC);
3             V_VAL_DESC[DBG$B_DHDR_TYPE] = DBG$K_V_VALUE_DESC;
3             V_VAL_DESC[DBG$W_DHDR_LENGTH] = 4 * (DBG$K_VALDESC_BASE_SIZE+4);
3             CH$MOVE (12, .PARAM4, V_VAL_DESC[DBG$A_VALUE_VMSDESC]);
3
3             ! Call the EVAL_LANG_OPERATOR routine to do the conversion.
3             !
3             DBG$EVAL_LANG_OPERATOR (
3                         DBG$GL_CONVERT_TOKEN,
3                         .VALUE_DESC,
3                         .V_VAL_DESC);
2             END;
2
2         ! Another case is during the output of watchpoints in
2         ! "old value", "new value".
2         !
2         [DBG$K_EXTERNAL_DESC] :
3             BEGIN
3             MAP
```

```
;  541          0669  3                     VALUE_DESC: REF DBG$PRIMARY;
;  542          0670  3
;  543          0671  3                 ! Check for aggregate.
;  544          0672  3                 !
;  545          0673  3                 IF .VALUE_DESC [DBG$V_DHDR_AGGR]
;  546          0674  3                 THEN
;  547          0675  3                     DBG$PRINT_AGGREGATE (.VALUE_DESC, .PARAM2)
;  548          0676  3
;  549          0677  3                 ELSE
;  550          0678  3
;  551          0679  3                     ! Call the PRINT_VALUE routine
;  552          0680  3                     !
;  553          0681  3                     DBG$PRINT_VALUE (.VALUE_DESC, .PARAM2, FALSE, FALSE);
;  554          0682  3
;  555          0683  3                 ! This is kind of a kludge. We fill in a -1 to PARAM5
;  556          0684  3                 ! and this indicates to the caller in DBGEXC that the
;  557          0685  3                 ! value has already been displayed.
;  558          0686  3                 !
;  559          0687  3                 .PARAM5 = -1;
;  560          0688  2                 END;
;  561          0689  2
;  562          0690  2             ! I don't think there are any other cases where this routine
;  563          0691  2             ! is still used, so signal an internal DEBUG error.
;  564          0692  2             !
;  565          0693  2             [OTHERWISE] :
;  566          0694  2                 $DBG_ERROR ('DBGLANVEC\DBG$NTYPE_CONV');
;  567          0695  2
;  568          0696  2             TES;
;  569          0697  2         RETURN STS$K_SUCCESS;
;  570          0698  1         END;



                                                                        .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

24  47  42  44  5C  43  45  56  4E  41  4C  47  42  44  18  0006E P.AAC:  .ASCII  <24>\DBGLANVEC\<92>\DBG$NTYPE_CONV\              ;
                    56  4E  4F  43  5F  45  50  59  54  4E  0007D                                                                   ;



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

                                                   007C 00000           .ENTRY  DBG$NTYPE_CONV, Save R2,R3,R4,R5,R6             ; 0498
                                50        0C    AC  D0 00002            MOVL    PARAM3, R0                                      ; 0632
                     00000082   8F              50  D1 00006            CMPL    R0, #130                                        ; 0641
                                                33  12 0000D            BNEQ    1$                                              ;
                                                0C  DD 0000F            PUSHL   #12                                             ; 0649
                     00000000G  00              01  FB 00011            CALLS   #1, DBG$GET_TEMPMEM                             ;
                                56              50  D0 00018            MOVL    R0, V_VAL_DESC                                  ;
                 66         04  BC              0C  28 0001B            MOVC3   #12, @VALUE_DESC, (V_VAL_DESC)                  ; 0650
                            02  A6        83    8F  90 00020            MOVB    #-125, 2(V_VAL_DESC)                            ; 0651
                                66              30  B0 00025            MOVW    #48, (V_VAL_DESC)                               ; 0652
            14   A6         10  BC              0C  28 00028            MOVC3   #12, @PARAM4, 20(V_VAL_DESC)                    ; 0653
                                                56  DD 0002E            PUSHL   V_VAL_DESC                                      ; 0660
                                          04    AC  DD 00030            PUSHL   VALUE_DESC                                      ; 0659
                                   00000000G    00  9F 00033            PUSHAB  DBG$GL_CONVERT_TOKEN                            ; 0657
                     00000000G  00              03  FB 00039            CALLS   #3, DBG$EVAL_LANG_OPERATOR                      ;
```

```
                                   48  11 00040       BRB     5$                              ; 0632
          00000081   8F            50  D1 00042 1$:   CMPL    R0, #129                        ; 0666
                                   2A  12 00049       BNEQ    4$                              ;
                     52     04     AC  D0 0004B       MOVL    VALUE_DESC, R2                  ; 0673
                     0E     04     A2  E9 0004F       BLBC    4(R2), 2$                       ;
                            08     AC  DD 00053       PUSHL   PARAM2                          ; 0675
                                   52  DD 00056       PUSHL   R2                              ;
          00000000G  00            02  FB 00058       CALLS   #2, DBG$PRINT_AGGREGATE         ;
                                   0E  11 0005F       BRB     3$                              ;
                                   7E  7C 00061 2$:   CLRQ    -(SP)                           ; 0681
                            08     AC  DD 00063       PUSHL   PARAM2                          ;
                                   52  DD 00066       PUSHL   R2                              ;
          00000000G  00            04  FB 00068       CALLS   #4, DBG$PRINT_VALUE             ;
                 14  BC            01  CE 0006F 3$:   MNEGL   #1, @PARAM5                     ; 0687
                                   15  11 00073       BRB     5$                              ; 0632
                      00000000'    EF  9F 00075 4$:   PUSHAB  P.AAC                           ; 0694
                                   01  DD 0007B       PUSHL   #1                              ;
                      00028362     8F  DD 0007D       PUSHL   #164706                         ;
          00000000G  00            03  FB 00083       CALLS   #3, LIB$SIGNAL                  ;
                     50            01  D0 0008A 5$:   MOVL    #1, R0                          ; 0697
                                       04 0008D       RET                                     ; 0698

; Routine Size:  142 bytes,    Routine Base:  DBG$CODE + 00B3
;  572          0699  1 GLOBAL ROUTINE DBG$NSYMBOLIZE (PRIM_DESC, PARAM2, PARAM3) =
;  573          0700  1 !
;  574          0701  1 ! FUNCTION
;  575          0702  1 !       Prints the name given by the primary descriptor, in the
;  576          0703  1 !       appropriate language format. This routine actually just
;  577          0704  1 !       passes the descriptor along to the new routine
;  578          0705  1 !       DBG$PRINT_IDENTIFIER.
;  579          0706  1 !
;  580          0707  1 ! FORMAL PARAMETERS:
;  581          0708  1 !
;  582          0709  1 !       PRIM_DESC       - A longword containing the address of a language specific
;  583          0710  1 !                         primary descriptor
;  584          0711  1 !
;  585          0712  1 !       PARAM2, PARAM3  - Unknown to this routine
;  586          0713  1 !
;  587          0714  1 ! IMPLICIT INPUTS:
;  588          0715  1 !
;  589          0716  1 !       NONE
;  590          0717  1 !
;  591          0718  1 ! IMPLICIT OUTPUTS:
;  592          0719  1 !
;  593          0720  1 !       Same as the invoked routine
;  594          0721  1 !
;  595          0722  1 ! ROUTINE VALUE:
;  596          0723  1 !
;  597          0724  1 !       Same as the invoked routine
;  598          0725  1 !
;  599          0726  1 ! COMPLETION CODES:
;  600          0727  1 !
;  601          0728  1 !       Same as the invoked routine
;  602          0729  1 !
;  603          0730  1 ! SIDE EFFECTS:
;  604          0731  1 !
;  605          0732  1 !       Same as the invoked routine.
;  606          0733  1 !
;  607          0734  1 !       This routine will generate a SIGNAL upon detection of a foreign
;  608          0735  1 !       language value within the primary descriptor.
;  609          0736  1 !
;  610          0737  2   BEGIN
;  611          0738  2   DBG$PRINT_IDENTIFIER (.PRIM_DESC);
;  612          0739  2   RETURN STS$K_SUCCESS;
;  613          0740  1   END;



                                       0000 00000          .ENTRY  DBG$NSYMBOLIZE, Save nothing              ; 0699
                                 04  AC  DD 00002          PUSHL   PRIM_DESC                                 ; 0738
              00000000G  00          01  FB 00005          CALLS   #1, DBG$PRINT_IDENTIFIER                  ;
                         50          01  D0 0000C          MOVL    #1, R0                                    ; 0739
                                         04 0000F          RET                                               ; 0740

; Routine Size:  16 bytes,    Routine Base:  DBG$CODE + 0141
```

```
GLOBAL ROUTINE DBG$NGET_PAGES (PRIM_DESC, PARAM2, PARAM3) =

!++
! FUNCTIONAL DESCRIPTION:
!
!       Uses a symbol's primary descriptor to construct a linked list of page
!       numbers which reflect those pages of storage in which the symbol's
!       rvalue is contained. Note that the pages may be non-contiguous.
!
!       A page number is represented by the high order 23 bits of a virtual
!       address, with the low order 9 bits set to 0:
!
!       page = (virtual__address AND B'11111111111111111111111000000000')
!
!       At implementation level 2,
!       This routine calls a language-specific routine depending on the language
!       code in the header of the descriptor.
!
!       At implementation level 3, the descriptors are the same so the
!       work is done right here.
!
! FORMAL PARAMETERS:
!
!       prim_desc       - A longword containing the address of a primary descriptor
!
!       param2          - The address of a longword to contain the address of the
!                         head node in the page list. Nodes in the page list
!                         consist of blocks of two longwords each. The second
!                         longword of the node block contains a page number on
!                         which some portion of the symbol's rvalue resides. The
!                         first longword of the node block contains the address
!                         of the next node in the list. The last node in the list
!                         should contain a 0 in this link field.
!
!       param3          - The address of a longword to contain the address of
!                         a message argument vector as described on page 4-119
!                         of the VAX/VMS system reference, volume 1A
!
! IMPLICIT INPUTS:
!
!       NONE
!
! IMPLICIT OUTPUTS:
!
!       In case of a success return, the page list is constructed from dynamic
!       storage and returned.
!
!       In case of a severe error return, a message arguement vector is constructed
!       and returned.
!
! ROUTINE VALUE:
!
!       An unsigned integer longword completion code
!
! COMPLETION CODES:
!
!       STS$K_SUCCESS (1) - Success. Page list constructed and returned.
```

```
!
!       STS$K_SEVERE  (4) - Failure. Page list not constructed. Message argument
!                           vector constructed and returned.
!
! SIDE EFFECTS:
!
!       NONE
!
    BEGIN

    MAP
        PRIM_DESC: REF DBG$PRIMARY;

    LOCAL
        BIT_LENGTH,                             ! Length of data in bits
        CURRENT_BLOCK: REF DBG$LINK_NODE,       ! Pointer to current page
                                                !    number block
        CURRENT_PAGE_ADDRESS,                   ! A page address
        END_ADDRESS,                            ! Last page address
        NEXT_BLOCK: REF DBG$LINK_NODE,          ! Pointer to the next page
                                                !    number block
        POS,
        VMS_DESC: REF DBG$STG_DESC,
        VMS_DESC_AREA: DBG$STG_DESC;

    ! For volatile value descriptors we already have a vms desc.
    !
    IF .PRIM_DESC [DBG$B_DHDR_TYPE] EQL DBG$K_V_VALUE_DESC
    OR .PRIM_DESC [DBG$B_DHDR_TYPE] EQL DBG$K_VALUE_DESC
    THEN
        VMS_DESC = PRIM_DESC [DBG$A_VALUE_VMSDESC]
    ELSE IF .PRIM_DESC [DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC
    THEN
        BEGIN

        ! Turn the primary descriptor into a VMS descriptor.
        !
        VMS_DESC = VMS_DESC_AREA;
        IF NOT DBG$MAKE_VMS_DESC (.PRIM_DESC, .VMS_DESC)
        THEN
            $DBG_ERROR ('DBGLANVEC\DBG$NGET_PAGES');
        END
    ELSE
        $DBG_ERROR ('DBGLANVEC\DBG$NGET_PAGES');

    ! The first address is given in the VMS descriptor. The end address
    ! must be computed from the bit length and the bit offset.
    !
    CURRENT_PAGE_ADDRESS = .VMS_DESC[DSC$A_POINTER] AND %X'FFFFFE00';
    BIT_LENGTH = DBG$DATA_LENGTH (.VMS_DESC);
    IF .VMS_DESC[DSC$B_CLASS] EQL DSC$K_CLASS_UBS
    THEN
        POS = .VMS_DESC[DSC$L_POS]
    ELSE
        POS = 0;
    END_ADDRESS = .VMS_DESC[DSC$A_POINTER] + (.BIT_LENGTH + .POS - 1)/8;
```

```
;  729          0855  2
;  730          0856  2      ! Loop through the pages.
;  731          0857  2      !
;  732          0858  2      CURRENT_BLOCK = 0;
;  733          0859  2      WHILE .CURRENT_PAGE_ADDRESS LEQ .END_ADDRESS DO
;  734          0860  3          BEGIN
;  735          0861  3
;  736          0862  3          ! Allocate space for a new node. Fill in the value field
;  737          0863  3          ! and link it in to the list (the list is actually being
;  738          0864  3          ! constructed backwards). Increment CURRENT_PAGE_ADDRESS to
;  739          0865  3          ! the next page and loop.
;  740          0866  3          !
;  741          0867  3          NEXT_BLOCK = DBG$GET_TEMPMEM (DBG$K_LINK_NODE_SIZE);
;  742          0868  3          NEXT_BLOCK[DBG$L_LINK_NODE_LINK] = .CURRENT_BLOCK;
;  743          0869  3          NEXT_BLOCK[DBG$L_LINK_NODE_VALUE] = .CURRENT_PAGE_ADDRESS;
;  744          0870  3          CURRENT_BLOCK = .NEXT_BLOCK;
;  745          0871  3          CURRENT_PAGE_ADDRESS = .CURRENT_PAGE_ADDRESS + 512;
;  746          0872  2          END;
;  747          0873  2
;  748          0874  2      ! Return the address of the last block.
;  749          0875  2      !
;  750          0876  2      .PARAM2 = .CURRENT_BLOCK;
;  751          0877  2      RETURN STS$K_SUCCESS;
;  752          0878  1      END;



                                                                        .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

24 47 42 44 5C 43 45 56 4E 41 4C 47 42 44 18 00087 P.AAD:  .ASCII  <24>\DBGLANVEC\<92>\DBG$NGET_PAGES\     ;
               53 45 47 41 50 5F 54 45 47 4E 00096                                                      ;
24 47 42 44 5C 43 45 56 4E 41 4C 47 42 44 18 000A0 P.AAE:  .ASCII  <24>\DBGLANVEC\<92>\DBG$NGET_PAGES\     ;
               53 45 47 41 50 5F 54 45 47 4E 000AF                                                      ;



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

                                                   001C 00000           .ENTRY  DBG$NGET_PAGES, Save R2,R3,R4     ; 0741
                              5E              0C   C2 00002           SUBL2   #12, SP
00000083   8F     04   BC     08              10   ED 00005           CMPZV   #16, #8, @PRIM_DESC, #131        ; 0825
                                              0C   13 0000F           BEQL    1$
0000007A   8F     04   BC     08              10   ED 00011           CMPZV   #16, #8, @PRIM_DESC, #122        ; 0826
                                              07   12 0001B           BNEQ    2$
                  52      04  AC              14   C1 0001D  1$:      ADDL3   #20, PRIM_DESC, VMS_DESC          ; 0828
                                              3B   11 00022           BRB     5$
00000079   8F     04   BC     08              10   ED 00024  2$:      CMPZV   #16, #8, @PRIM_DESC, #121        ; 0829
                                              1A   12 0002E           BNEQ    3$
                              52              6E   9E 00030           MOVAB   VMS_DESC_AREA, VMS_DESC           ; 0835
                                              52   DD 00033           PUSHL   VMS_DESC                          ; 0836
                                     04       AC   DD 00035           PUSHL   PRIM_DESC                         ;
                  00000000G   00              02   FB 00038           CALLS   #2, DBG$MAKE_VMS_DESC             ;
                              1D              50   E8 0003F           BLBS    R0, 5$                            ;
                                 00000000'    EF   9F 00042           PUSHAB  P.AAD                             ; 0838
                                              06   11 00048           BRB     4$                                ;
                                 00000000'    EF   9F 0004A  3$:      PUSHAB  P.AAE                             ; 0841
                                              01   DD 00050  4$:      PUSHL   #1                                ;
```

```
                          00028362   8F  DD 00052       PUSHL   #164706                                            ;
             00000000G  00            03  FB 00058       CALLS   #3, LIB$SIGNAL                                     ;
       54         04    A2 000001FF   8F  CB 0005F  5$:  BICL3   #511, 4(VMS_DESC), CURRENT_PAGE_ADDRESS            ; 0846
                                      52  DD 00068       PUSHL   VMS_DESC                                           ; 0847
             00000000G  00            01  FB 0006A       CALLS   #1, DBG$DATA_LENGTH                                ;
                        0D        03  A2  91 00071       CMPB    3(VMS_DESC), #13                                   ; 0848
                                      06  12 00075       BNEQ    6$                                                 ;
                        51        08  A2  D0 00077       MOVL    8(VMS_DESC), POS                                   ; 0850
                                      02  11 0007B       BRB     7$                                                 ;
                                      51  D4 0007D  6$:  CLRL    POS                                                ; 0852
                        50     FF A140  9E 0007F  7$:  MOVAB   -1(POS)[BIT_LENGTH], R0                            ; 0853
                        50            08  C6 00084       DIVL2   #8, R0                                             ;
       53               50        04  A2  C1 00087       ADDL3   4(VMS_DESC), R0, END_ADDRESS                       ;
                                      52  D4 0008C       CLRL    CURRENT_BLOCK                                      ; 0858
                        53            54  D1 0008E  8$:  CMPL    CURRENT_PAGE_ADDRESS, END_ADDRESS                  ; 0859
                                      1A  14 00091       BGTR    9$                                                 ;
                                      02  DD 00093       PUSHL   #2                                                 ; 0867
             00000000G  00            01  FB 00095       CALLS   #1, DBG$GET_TEMPMEM                                ;
                        60            52  D0 0009C       MOVL    CURRENT_BLOCK, (NEXT_BLOCK)                        ; 0868
                  04    A0            84  7E 0009F       MOVAQ   (CURRENT_PAGE_ADDRESS)+, 4(NEXT_BLOCK)             ; 0869
                        52            50  D0 000A3       MOVL    NEXT_BLOCK, CURRENT_BLOCK                          ; 0870
                        54      01F8  C4  9E 000A6       MOVAB   504(R4), CURRENT_PAGE_ADDRESS                      ; 0871
                                      E1  11 000AB       BRB     8$                                                 ; 0859
                  08    BC            52  D0 000AD  9$:  MOVL    CURRENT_BLOCK, @PARAM2                             ; 0876
                        50            01  D0 000B1       MOVL    #1, R0                                             ; 0877
                                          04 000B4       RET                                                        ; 0878

; Routine Size:  181 bytes,    Routine Base:  DBG$CODE + 0151
```

```
GLOBAL ROUTINE DBG$NGET_LENGTH (PRIM_DESC, PARAM2, PARAM3) =
!++
! FUNCTIONAL DESCRIPTION:
!
!       Uses a symbol's primary descriptor to obtain the length of the symbol's
!       rvalue. The length is to be given in bits. Lengths longer than 2 ** 32
!       must be truncated to this length.
!
!       The debugger assumes that rvalues refer to contiguous blocks of storage.
!       If this is not true for a given variable, this routine fails.
!
!       Length should reflect the maximum length for entities that may vary in
!       size, and include the length of a control word, if one is present.
!
!       If the value of the object can not be materialized by the Type Convertor
!       (DBG$NTYPE_CONV), this routine should return STS$K_INFO. This is
!       generally true for objects of aggregate type, e.g., PASCAL arrays and
!       record, PL/I structures.
!
!       This routine calls a language-specific routine based on the language
!       code in the descriptor header.
!
! FORMAL PARAMETERS:
!
!       prim_desc         - A longword containing the address of a primary descriptor
!
!       param2            - The address of a longword to contain an unsigned integer
!                           longword representing the symbol's rvalue length in bits
!
!       param3            - The address of a longword to contain the address of a
!                           message argument vector as described on page 4-119 of
!                           the VAX/VMS system reference, volume 1A
!
! IMPLICIT INPUTS:
!
!       NONE
!
! IMPLICIT OUTPUTS:
!
!       In case of a severe error return, a message argument vector is constructed
!       from dynamic storage and returned.
!
! ROUTINE VALUE:
!
!       An unsigned integer longword completion code
!
! COMPLETION CODES:
!
!       STS$K_SUCCESS (1) - Success. Length of symbol's rvalue returned.
!
!       STS$K_INFO    (3) - Success. Length of the symbol's rvalue returned but
!                           the symbol refers to a value that the Type Convertor
!                           cannot materialize.
!
!       STS$K_SEVERE  (4) - Failure. No length returned. Message argument vector
!                           constructed and returned.
!
```

```
;  811         0936   1 ! SIDE EFFECTS:
;  812         0937   1 !
;  813         0938   1 !       NONE
;  814         0939   1 !
;  815         0940   1 !
;  816         0941   2     BEGIN
;  817         0942   2
;  818         0943   2     MAP
;  819         0944   2         PRIM_DESC: REF DBG$VALDESC;
;  820         0945   2     LOCAL
;  821         0946   2         VMS_DESC,
;  822         0947   2         VMS_DESC_AREA: DBG$STG_DESC;
;  823         0948   2
;  824         0949   2     ! Primary Descriptors.
;  825         0950   2     !
;  826         0951   2     IF .PRIM_DESC [DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC
;  827         0952   2     THEN
;  828         0953   3         BEGIN
;  829         0954   3
;  830         0955   3         ! Call a routine to construct the VMS descriptor.
;  831         0956   3         !
;  832         0957   3         VMS_DESC = VMS_DESC_AREA;
;  833         0958   3         IF NOT DBG$MAKE_VMS_DESC (.PRIM_DESC, .VMS_DESC)
;  834         0959   3         THEN
;  835         0960   3             $DBG_ERROR ('DBGLANVEC\DBG$NGET_LENGTH');
;  836         0961   3         END
;  837         0962   3
;  838         0963   3     ! Volatile Value Descriptors or Value Descriptors.
;  839         0964   3     !
;  840         0965   2     ELSE IF .PRIM_DESC [DBG$B_DHDR_TYPE] EQL DBG$K_V_VALUE_DESC
;  841         0966   2     OR .PRIM_DESC [DBG$B_DHDR_TYPE] EQL DBG$K_VALUE_DESC
;  842         0967   2     THEN
;  843         0968   2
;  844         0969   2         ! In this case just get the VMS descriptor out of the
;  845         0970   2         ! volatile value descriptor.
;  846         0971   2         !
;  847         0972   2         VMS_DESC = PRIM_DESC [DBG$A_VALUE_VMSDESC]
;  848         0973   2
;  849         0974   2     ! We do not expect any other kind of descriptor.
;  850         0975   2     !
;  851         0976   2     ELSE
;  852         0977   2         $DBG_ERROR ('DBGLANVEC\DBG$NGET_LENGTH unknown descriptor type');
;  853         0978   2
;  854         0979   2
;  855         0980   2     ! Call the routine in DBGVALUES that extracts a bit length from
;  856         0981   2     ! a VMS descriptor.
;  857         0982   2     !
;  858         0983   2     .PARAM2 = DBG$DATA_LENGTH (.VMS_DESC);
;  859         0984   2     RETURN STS$K_SUCCESS;
;  860         0985   1     END;


                                                                  .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

24  47  42  44  5C  43  45  56  4E  41  4C  47  42  44  19  000B9 P.AAF:  .ASCII  <25>\DBGLANVEC\<92>\DBG$NGET_LENGTH\         ;
                48  54  47  4E  45  4C  5F  54  45  47  4E  000C8                                                               ;
```

```
                                                                     N 14
DBGLANVEC                                                            16-Sep-1984 01:24:56     VAX-11 Bliss-32 V4.0-742              Page  24
V04-000                                                              14-Sep-1984 12:17:01     [DEBUG.SRC]DBGLANVEC.B32;1                  (9)

24  47  42  44  5C  43  45  56  4E  41  4C  47  42  44  31  000D3 P.AAG:  .ASCII  \1DBGLANVEC\<92>\DBG$NGET_LENGTH unknown\  ;
6B  6E  75  20  48  54  47  4E  45  4C  5F  54  45  47  4E  000E2                                                          ;
                                            6E  77  6F  6E  000F1                                                          ;
70  79  74  20  72  6F  74  70  69  72  63  73  65  64  20  000F5         .ASCII  \ descriptor type\                       ;
                                                        65  00104                                                          ;


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

                                                          0004 00000      .ENTRY  DBG$NGET_LENGTH, Save R2                 ; 0879
                                 5E                   0C  C2 00002        SUBL2   #12, SP                                  ;
00000079    8F       04    BC    08                   10  ED 00005        CMPZV   #16, #8, @PRIM_DESC, #121                ; 0951
                                                      1A  12 0000F        BNEQ    1$                                       ;
                                 52                   6E  9E 00011        MOVAB   VMS_DESC_AREA, VMS_DESC                  ; 0957
                                                      52  DD 00014        PUSHL   VMS_DESC                                 ; 0958
                                             04       AC  DD 00016        PUSHL   PRIM_DESC                                ;
                         00000000G  00                02  FB 00019        CALLS   #2, DBG$MAKE_VMS_DESC                    ;
                                    3C                50  E8 00020        BLBS    R0, 5$                                   ;
                                       00000000'      EF  9F 00023        PUSHAB  P.AAF                                    ; 0960
                                                      25  11 00029        BRB     4$                                       ;
00000083    8F       04    BC       08                10  ED 0002B 1$:    CMPZV   #16, #8, @PRIM_DESC, #131                ; 0965
                                                      0C  13 00035        BEQL    2$                                       ;
0000007A    8F       04    BC       08                10  ED 00037        CMPZV   #16, #8, @PRIM_DESC, #122                ; 0966
                                                      07  12 00041        BNEQ    3$                                       ;
                           52    04     AC            14  C1 00043 2$:    ADDL3   #20, PRIM_DESC, VMS_DESC                 ; 0972
                                                      15  11 00048        BRB     5$                                       ;
                                       00000000'      EF  9F 0004A 3$:    PUSHAB  P.AAG                                    ; 0977
                                                      01  DD 00050 4$:    PUSHL   #1                                       ;
                                       00028362       8F  DD 00052        PUSHL   #164706                                  ;
                         00000000G  00                03  FB 00058        CALLS   #3, LIB$SIGNAL                           ;
                                                      52  DD 0005F 5$:    PUSHL   VMS_DESC                                 ; 0983
                         00000000G  00                01  FB 00061        CALLS   #1, DBG$DATA_LENGTH                      ;
                                08  BC                50  D0 00068        MOVL    R0, @PARAM2                              ;
                                    50                01  D0 0006C        MOVL    #1, R0                                   ; 0984
                                                      04     0006F        RET                                              ; 0985

; Routine Size:  112 bytes,    Routine Base:  DBG$CODE + 0206
```

```
1 GLOBAL ROUTINE DBG$NCOPY_DESC (DESC, PARAM2, PARAM3, PARAM4) =
1 !
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       Accepts as input a language specific primary or value descriptor
1 !       (constructed from listed storage)
1 !       and makes a copy of the descriptor out of non-listed storage. This
1 !       non-volatile copy will be stored in conjunction with x-points and
1 !       current location.
1 !
1 !       This routine may use DBG$NCOPY to copy each portion of the
1 !       descriptor that has been created from listed dynamic storage.
1 !
1 ! FORMAL PARAMETERS:
1 !
1 !       desc                        - The address of a language specifc primary or
1 !                                     value descriptor
1 !
1 !       param2                      - The address of a longword to contain the address
1 !                                     of the non-volatile copy of the descriptor
1 !
1 !       param3                      - The address of a longword to contain the address
1 !                                     of a message argument vector for errors
1 !
1 !       param4                      - A flag saying whether to copy into permanent
1 !                                     memory or temporary memory. Only used in
1 !                                     implementation level 3.
1 ! IMPLICIT INPUTS:
1 !
1 !       NONE
1 !
1 ! IMPLICIT OUTPUTS:
1 !
1 !       On success, the non-volatile copy of a primary descriptor.
1 !
1 !       On failure, a message argument vector.
1 !
1 ! ROUTINE VALUE:
1 !
1 !       An unsigned integer longword completion code
1 !
1 ! COMPLETION CODES:
1 !
1 !       STS$K_SUCCESS    (1)        - Success. Copy constructed and returned.
1 !
1 !       STS$K_SEVERE     (4)        - Failure. Copy not produced. Message argument
1 !                                     vector constructed and returned.
1 !
1 ! SIDE EFFECTS:
1 !
1 !       NONE
1 !
2     BEGIN
2
2     MAP
2       DESC: REF DBG$VALDESC;
2
```

```
BUILTIN
    ACTUALCOUNT;                ! Count of acual parameters.

LOCAL
    LENGTH,                     ! Length in bytes of copy
    PERM_FLAG;                  ! Flag saying whether to copy into permanent
                                !     or temporary memory.

! Enable a handler which will take care of NOFREE error messages.
! The reason for this is, if we run out of memory part way through
! copying the descriptor, then we want to release the memory we
! have allocated so far, so that it does not get lost forever.
!
ENABLE
    COPY_DESC_HANDLER;

! Default the fourth parameter to TRUE.
! Also initialize the pointer to the new descriptor header.
!
IF ACTUALCOUNT() LSS 4
THEN
    PERM_FLAG = TRUE
ELSE
    PERM_FLAG = .PARAM4;
COPY_DESC_HEAD = 0;

! Compute the number of bytes to allocate. Always allocate
! at least 16 + base size of value descriptor.
!
LENGTH = .DESC[DBG$W_DHDR_LENGTH];
IF .LENGTH LSS 16 + 4*DBG$K_VALDESC_BASE_SIZE
THEN
    LENGTH = 16 + 4*DBG$K_VALDESC_BASE_SIZE;

CASE .DESC [DBG$B_DHDR_TYPE] FROM DBG$K_LITERAL TO DBG$K_V_VALUE_DESC OF
    SET

    ! Ordinary value descriptors. These have the actual value embedded
    ! inside them. Copy the descriptor and fix up the pointer field
    ! so it points to the right place.
    !
    [DBG$K_VALUE_DESC]:
        BEGIN
        MAP
            DESC: REF DBG$VALDESC;  ! Pointer to a new style value
                                    !     descriptor (the original)
        LOCAL
            DESC_COPY: REF DBG$VALDESC;     ! Pointer to a new style value
                                            !     descriptor (the copy).

        IF .PERM_FLAG
        THEN
            DESC_COPY = DBG$GET_MEMORY ((3+.LENGTH)/4)
        ELSE
            DESC_COPY = DBG$GET_TEMPMEM ((3+.LENGTH)/4);
        CH$MOVE (.DESC[DBG$W_DHDR_LENGTH], .DESC, .DESC_COPY);
        DESC_COPY [DBG$L_VALUE_POINTER] = DESC_COPY [DBG$A_VALUE_ADDRESS];
```

```
3                  .PARAM2 = .DESC_COPY;
2                  END;
2
2          ! Volatile value descriptors. These point to a region of user
2          ! memory containing the value. We do the same as above except
2          ! that we do not fix up the pointer field.
2          !
2          [DBG$K_V_VALUE_DESC]:
3                  BEGIN
3                  MAP
3                      DESC: REF DBG$VALDESC;  ! Pointer to a new style value
3                                              !          descriptor (the original)
3                  LOCAL
3                      DESC_COPY: REF DBG$VALDESC;       ! Pointer to a new style value
3                                                        !          descriptor (the copy).
3
3                  IF .PERM_FLAG
3                  THEN
3                      DESC_COPY = DBG$GET_MEMORY ((3+.LENGTH)/4)
3                  ELSE
3                      DESC_COPY = DBG$GET_TEMPMEM ((3+.LENGTH)/4);
3                  CH$MOVE (.DESC[DBG$W_DHDR_LENGTH], .DESC, .DESC_COPY);
2                  .PARAM2 = .DESC_COPY;
2                  END;
2
2          ! New style Primary Descriptors. Here we have to copy the root
2          ! node and all sub-nodes. Note that we have to do this carefully,
2          ! in such as way that at any time we call GET_MEMORY, we must
2          ! have a valid (though partially constructed) Primary. This is
2          ! in case GET_MEMORY signals a NOFREE error message - we want
2          ! to be able to release the storage we have allocated up
2          ! to the point of running out of memory.
2          !
2          [DBG$K_PRIMARY_DESC]:
3                  BEGIN
3
3                  MAP
3                      DESC: REF DBG$PRIMARY;                     ! Pointer to the Primary
3                                                                 !      Descriptor to
3                                                                 !      be copied.
3                  LOCAL
3                      DESC_COPY : REF DBG$PRIMARY,               ! Pointer to the copy
3                                                                 !      of the Primary
3                                                                 !      Descriptor.
3                      DIMCNT,
3                      NEW_SUBNODE: REF DBG$PRIM_NODE,            ! Pointer to a copy of
3                                                                 !    a subnode
3                      NUMBLKS,
3                      PREV_SUBNODE: REF DBG$PRIM_NODE,           ! Pointer to a copy of
3                                                                 !    a subnode
3                      SIZE,                                      ! Size of a subnode
3                      SUBCNT,
3                      SUBNODE: REF DBG$PRIM_NODE;                ! Pointer to the original
3                                                                 !    subnode.
3
3                  ! Allocate memory for a new root node and copy the
3                  ! values into it. We will fix up forward and back
```

```
3   ! links later.
3   !
3   IF .PERM_FLAG
3   THEN
4       BEGIN
4       DESC_COPY = DBG$GET_MEMORY (DBG$K_PRIMARY_SIZE);
4
4       ! Put a pointer to the Primary in this own variable so
4       ! COPY_DESC_HANDLER can later free up the storage.
4       !
4       COPY_DESC_HEAD = .DESC_COPY;
4       END
3   ELSE
3       DESC_COPY = DBG$GET_TEMPMEM (DBG$K_PRIMARY_SIZE);
3   CH$MOVE (4*DBG$K_PRIMARY_SIZE, .DESC, .DESC_COPY);
3
3   ! Fix up the forward and back links so we have a valid partially
3   ! constructed Primary - i.e., we do not want to leave them pointing
3   ! to the original Primary.
3   !
3   DESC_COPY[DBG$L_PRIM_FLINK] = DESC_COPY[DBG$L_PRIM_FLINK];
3   DESC_COPY[DBG$L_PRIM_BLINK] = DESC_COPY[DBG$L_PRIM_FLINK];
3
3   ! Loop through each of the subnodes.
3   !
3   SUBNODE = .DESC [DBG$L_PRIM_FLINK];
3   PREV_SUBNODE = 0;
3   WHILE .SUBNODE NEQ DESC[DBG$L_PRIM_FLINK] DO
4       BEGIN
4
4       ! Allocate space for the new subnode.
4       !
4       IF .SUBNODE [DBG$B_PNODE_FCODE] EQL RST$K_TYPE_ARRAY
4       THEN
5           BEGIN
5           ! Use larger of SUBCNT, DIMCNT.
5           SUBCNT = .SUBNODE[DBG$B_PNARR_SUBCNT];
5           DIMCNT = .SUBNODE[DBG$B_PNARR_DIMCNT];
5           IF .SUBCNT GTR .DIMCNT
5           THEN
5               NUMBLKS = .SUBCNT
5           ELSE
5               NUMBLKS = .DIMCNT;
5           SIZE = DBG$K_PRIM_SIZE_ARRAY +
5                       DBG$K_PRIM_SIZE_SUBS*.NUMBLKS;
5           END
4       ELSE IF .SUBNODE [DBG$B_PNODE_FCODE] EQL RST$K_TYPE_RECORD
4       THEN
4           SIZE = DBG$K_PRIM_SIZE_RECORD
4       ELSE IF .SUBNODE [DBG$B_PNODE_FCODE] EQL RST$K_TYPE_VARIANT
4       THEN
4           SIZE = DBG$K_PRIM_SIZE_VARIANT
4       ELSE
4           SIZE = DBG$K_PRIM_SIZE_NORMAL;
4       IF .PERM_FLAG
4       THEN
4           NEW_SUBNODE = DBG$GET_MEMORY(.SIZE)
```

```
; 1090          1214  4                     ELSE
; 1091          1215  4                         NEW_SUBNODE = DBG$GET_TEMPMEM(.SIZE);
; 1092          1216  4
; 1093          1217  4                     ! Copy the values.
; 1094          1218  4                     !
; 1095          1219  4                     CH$MOVE (4*.SIZE, .SUBNODE, .NEW_SUBNODE);
; 1096          1220  4
; 1097          1221  5                     IF .PERM_FLAG AND (.SUBNODE [DBG$B_PNODE_FCODE] EQL RST$K_TYPE_VARIANT)
; 1098          1222  4                     THEN
; 1099          1223  4                         NEW_SUBNODE[DBG$V_PNVAR_VALID] = FALSE;
; 1100          1224  4
; 1101          1225  4                     ! Fill in the forward and back links.
; 1102          1226  4                     !
; 1103          1227  4                     IF .PREV_SUBNODE EQL 0
; 1104          1228  4                     THEN
; 1105          1229  5                         BEGIN
; 1106          1230  5                         DESC_COPY [DBG$L_PRIM_FLINK] = .NEW_SUBNODE;
; 1107          1231  5                         DESC_COPY [DBG$L_PRIM_BLINK] = .NEW_SUBNODE;
; 1108          1232  5                         NEW_SUBNODE [DBG$L_PNODE_FLINK] = DESC_COPY [DBG$A_PRIM_FLINK];
; 1109          1233  5                         NEW_SUBNODE [DBG$L_PNODE_BLINK] = DESC_COPY [DBG$A_PRIM_FLINK];
; 1110          1234  5                         END
; 1111          1235  4                     ELSE
; 1112          1236  5                         BEGIN
; 1113          1237  5                         PREV_SUBNODE [DBG$L_PNODE_FLINK] = .NEW_SUBNODE;
; 1114          1238  5                         DESC_COPY [DBG$L_PRIM_BLINK] = .NEW_SUBNODE;
; 1115          1239  5                         NEW_SUBNODE [DBG$L_PNODE_FLINK] = DESC_COPY [DBG$A_PRIM_FLINK];
; 1116          1240  5                         NEW_SUBNODE [DBG$L_PNODE_BLINK] = .PREV_SUBNODE;
; 1117          1241  4                         END;
; 1118          1242  4                     PREV_SUBNODE = .NEW_SUBNODE;
; 1119          1243  4                     SUBNODE = .SUBNODE [DBG$L_PNODE_FLINK];
; 1120          1244  3                     END;
; 1121          1245  3
; 1122          1246  3                 .PARAM2 = .DESC_COPY;
; 1123          1247  2                 END;
; 1124          1248  2
; 1125          1249  2             ! At implementation level 3, we do not expect any other kind
; 1126          1250  2             ! of descriptor.
; 1127          1251  2             !
; 1128          1252  2             [INRANGE, OUTRANGE]:
; 1129          1253  2                 $DBG_ERROR ('DBGLANVEC\DBG$NCOPY_DESC');
; 1130          1254  2
; 1131          1255  2             TES;
; 1132          1256  2
; 1133          1257  2         ! The copying has been done. Return success.
; 1134          1258  2         !
; 1135          1259  2         RETURN STS$K_SUCCESS;
; 1136          1260  1         END;



                                                                      .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

24  47  42  44  5C  43  45  56  4E  41  4C  47  42  44  18  00105 P.AAH:  .ASCII  <24>\DBGLANVEC\<92>\DBG$NCOPY_DESC\        ;
                    43  53  45  44  5F  59  50  4F  43  4E  00114                                                              ;
```

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

                                          0FFC 00000            .ENTRY  DBG$NCOPY_DESC, Save R2,R3,R4,R5,R6,R7,R8,-  ; 0986
                                                                        R9,R10,R11                                   ;
                              5E           14  C2 00002         SUBL2   #20, SP                                      ;
                              6D     01A7  CF  DE 00005         MOVAL   32$, (FP)                                    ; 1038
                              04           6C  91 0000A         CMPB    (AP), #4                                     ; 1062
                                           05  1E 0000D         BGEQU   1$                                           ;
                              6E           01  D0 0000F         MOVL    #1, PERM_FLAG                                ; 1064
                                           04  11 00012         BRB     2$                                           ;
                              6E       10  AC  D0 00014 1$:     MOVL    PARAM4, PERM_FLAG                            ; 1066
                                00000000'  EF  D4 00018 2$:     CLRL    COPY_DESC_HEAD                               ; 1067
                              5B       04  AC  D0 0001E         MOVL    DESC, R11                                    ; 1072
                              50           6B  3C 00022         MOVZWL  (R11), LENGTH                                ;
                              30           50  D1 00025         CMPL    LENGTH, #48                                  ; 1073
                                           03  18 00028         BGEQ    3$                                           ;
                              50           30  D0 0002A         MOVL    #48, LENGTH                                  ; 1075
            0B         78     8F       02  AB  8F 0002D 3$:     CASEB   2(R11), #120, #11                            ; 1077
 0018       002F              0085         0018   00033 4$:     .WORD   5$-4$,-                                      ;
 0018       0018              0018         0018   0003B                 14$-4$,-                                     ;
 005A       0018              0018         0018   00043                 6$-4$,-                                      ;
                                                                        5$-4$,-                                      ;
                                                                        5$-4$,-                                      ;
                                                                        5$-4$,-                                      ;
                                                                        5$-4$,-                                      ;
                                                                        5$-4$,-                                      ;
                                                                        5$-4$,-                                      ;
                                                                        5$-4$,-                                      ;
                                                                        5$-4$,-                                      ;
                                                                        9$-4$                                        ;
                                00000000'  EF  9F 0004B 5$:     PUSHAB  P.AAH                                        ; 1253
                                           01  DD 00051         PUSHL   #1                                           ;
                                00028362   8F  DD 00053         PUSHL   #164706                                      ;
                 00000000G    00           03  FB 00059         CALLS   #3, LIB$SIGNAL                               ;
                                           53  11 00060         BRB     13$                                          ;
                              50           03  C0 00062 6$:     ADDL2   #3, R0                                       ; 1095
                              50           04  C6 00065         DIVL2   #4, R0                                       ;
                              0B           6E  E9 00068         BLBC    PERM_FLAG, 7$                                ; 1093
                                           50  DD 0006B         PUSHL   R0                                           ; 1095
                 00000000G    00           01  FB 0006D         CALLS   #1, DBG$GET_MEMORY                           ;
                                           09  11 00074         BRB     8$                                           ;
                                           50  DD 00076 7$:     PUSHL   R0                                           ; 1097
                 00000000G    00           01  FB 00078         CALLS   #1, DBG$GET_TEMPMEM                          ;
                              56           50  D0 0007F 8$:     MOVL    R0, DESC_COPY                                ;
            66                6B           6B  28 00082         MOVC3   (R11), (R11), (DESC_COPY)                    ; 1098
                       18     A6       20  A6  9E 00086         MOVAB   32(R6), 24(DESC_COPY)                        ; 1099
                                           24  11 0008B         BRB     12$                                          ; 1100
                              50           03  C0 0008D 9$:     ADDL2   #3, R0                                       ; 1118
                              50           04  C6 00090         DIVL2   #4, R0                                       ;
                              0B           6E  E9 00093         BLBC    PERM_FLAG, 10$                               ; 1116
                                           50  DD 00096         PUSHL   R0                                           ; 1118
                 00000000G    00           01  FB 00098         CALLS   #1, DBG$GET_MEMORY                           ;
                                           09  11 0009F         BRB     11$                                          ;
                                           50  DD 000A1 10$:    PUSHL   R0                                           ; 1120
                 00000000G    00           01  FB 000A3         CALLS   #1, DBG$GET_TEMPMEM                          ;
                              56           50  D0 000AA 11$:    MOVL    R0, DESC_COPY                                ;
            66                6B           6B  28 000AD         MOVC3   (R11), (R11), (DESC_COPY)                    ; 1121
```

```
                08  BC            56  D0 000B1 12$:  MOVL    DESC_COPY, @PARAM2                        ; 1122
                                00F4  31 000B5 13$:  BRW     31$                                       ; 1077
                    15            6E  E9 000B8 14$:  BLBC    PERM_FLAG, 15$                            ; 1159
                                  09  DD 000BB       PUSHL   #9                                        ; 1162
         00000000G  00            01  FB 000BD       CALLS   #1, DBG$GET_MEMORY                        ;
                    5A            50  D0 000C4       MOVL    R0, DESC_COPY                             ;
         00000000'  EF            5A  D0 000C7       MOVL    DESC_COPY, COPY_DESC_HEAD                 ; 1167
                                  0C  11 000CE       BRB     16$                                       ; 1159
                                  09  DD 000D0 15$:  PUSHL   #9                                        ; 1170
         00000000G  00            01  FB 000D2       CALLS   #1, DBG$GET_TEMPMEM                       ;
                    5A            50  D0 000D9       MOVL    R0, DESC_COPY                             ;
    6A              6B            24  28 000DC 16$:  MOVC3   #36, (R11), (DESC_COPY)                   ; 1171
                08  AE      14    AA  9E 000E0       MOVAB   20(DESC_COPY), 8(SP)                      ; 1177
                08  BE      08    AE  D0 000E5       MOVL    8(SP), @8(SP)                             ;
                18  AA      08    AE  D0 000EA       MOVL    8(SP), 24(DESC_COPY)                      ; 1178
                    59      14    AB  D0 000EF       MOVL    20(R11), SUBNODE                          ; 1182
                            04    AE  D4 000F3       CLRL    PREV_SUBNODE                              ; 1183
                    50      14    AB  9E 000F6 17$:  MOVAB   20(R11), R0                               ; 1184
                    50            59  D1 000FA       CMPL    SUBNODE, R0                               ;
                                  03  12 000FD       BNEQ    18$                                       ;
                                00A6  31 000FF       BRW     30$                                       ;
                    01      09    A9  91 00102 18$:  CMPB    9(SUBNODE), #1                            ; 1189
                                  25  12 00106       BNEQ    21$                                       ;
                0C  AE      1F    A9  9A 00108       MOVZBL  31(SUBNODE), SUBCNT                       ; 1193
                10  AE      1B    A9  9A 0010D       MOVZBL  27(SUBNODE), DIMCNT                       ; 1194
                10  AE      0C    AE  D1 00112       CMPL    SUBCNT, DIMCNT                            ; 1195
                                  06  15 00117       BLEQ    19$                                       ;
                    57      0C    AE  D0 00119       MOVL    SUBCNT, NUMBLKS                           ; 1197
                                  04  11 0011D       BRB     20$                                       ;
                    57      10    AE  D0 0011F 19$:  MOVL    DIMCNT, NUMBLKS                           ; 1199
    50              57            05  C5 00123 20$:  MULL3   #5, NUMBLKS, R0                           ; 1201
                    56      0A    A0  9E 00127       MOVAB   10(R0), SIZE                              ; 1200
                                  19  11 0012B       BRB     24$                                       ; 1189
                    07      09    A9  91 0012D 21$:  CMPB    9(SUBNODE), #7                            ; 1203
                                  05  12 00131       BNEQ    22$                                       ;
                    56            07  D0 00133       MOVL    #7, SIZE                                  ; 1205
                                  0E  11 00136       BRB     24$                                       ;
                    13      09    A9  91 00138 22$:  CMPB    9(SUBNODE), #19                           ; 1206
                                  05  12 0013C       BNEQ    23$                                       ;
                    56            0A  D0 0013E       MOVL    #10, SIZE                                 ; 1208
                                  03  11 00141       BRB     24$                                       ;
                    56            06  D0 00143 23$:  MOVL    #6, SIZE                                  ; 1210
                    0B            6E  E9 00146 24$:  BLBC    PERM_FLAG, 25$                            ; 1213
                                  56  DD 00149       PUSHL   SIZE                                      ;
         00000000G  00            01  FB 0014B       CALLS   #1, DBG$GET_MEMORY                        ;
                                  09  11 00152       BRB     26$                                       ;
                                  56  DD 00154 25$:  PUSHL   SIZE                                      ; 1215
         00000000G  00            01  FB 00156       CALLS   #1, DBG$GET_TEMPMEM                       ;
                    58            50  D0 0015D 26$:  MOVL    R0, NEW_SUBNODE                           ;
    50              56            02  78 00160       ASHL    #2, SIZE, R0                              ; 1219
    68              69            50  28 00164       MOVC3   R0, (SUBNODE), (NEW_SUBNODE)              ;
                    0A            6E  E9 00168       BLBC    PERM_FLAG, 27$                            ; 1221
                    13      09    A9  91 0016B       CMPB    9(SUBNODE), #19                           ;
                                  04  12 0016F       BNEQ    27$                                       ;
                0A  A8            10  8A 00171       BICB2   #16, 10(NEW_SUBNODE)                      ; 1223
                            04    AE  D5 00175 27$:  TSTL    PREV_SUBNODE                              ; 1227
                                  13  12 00178       BNEQ    28$                                       ;
```

```
                 08    BE             58  D0 0017A          MOVL    NEW_SUBNODE, @8(SP)                      ; 1230
                 18    AA             58  D0 0017E          MOVL    NEW_SUBNODE, 24(DESC_COPY)               ; 1231
                       68     08      AE  D0 00182          MOVL    8(SP), (NEW_SUBNODE)                     ; 1232
                 04    A8     08      AE  D0 00186          MOVL    8(SP), 4(NEW_SUBNODE)                    ; 1233
                                      11  11 0018B          BRB     29$                                      ; 1227
                 04    BE             58  D0 0018D 28$:     MOVL    NEW_SUBNODE, @PREV_SUBNODE               ; 1237
                 18    AA             58  D0 00191          MOVL    NEW_SUBNODE, 24(DESC_COPY)               ; 1238
                       68     08      AE  D0 00195          MOVL    8(SP), (NEW_SUBNODE)                     ; 1239
                 04    A8     04      AE  D0 00199          MOVL    PREV_SUBNODE, 4(NEW_SUBNODE)             ; 1240
                 04    AE             58  D0 0019E 29$:     MOVL    NEW_SUBNODE, PREV_SUBNODE                ; 1242
                       59             69  D0 001A2          MOVL    (SUBNODE), SUBNODE                       ; 1243
                                    FF4E  31 001A5          BRW     17$                                      ; 1184
                 08    BC             5A  D0 001A8 30$:     MOVL    DESC_COPY, @PARAM2                       ; 1246
                       50             01  D0 001AC 31$:     MOVL    #1, R0                                   ; 1259
                                          04 001AF          RET                                              ; 1260
                                       0000 001B0 32$:      .WORD   Save nothing                             ; 1038
                                      7E  D4 001B2          CLRL    -(SP)                                    ;
                                      5E  DD 001B4          PUSHL   SP                                       ;
                       7E     04      AC  7D 001B6          MOVQ    4(AP), -(SP)                             ;
              0000V    CF             03  FB 001BA          CALLS   #3, COPY_DESC_HANDLER                    ;
                                          04 001BF          RET                                              ;

; Routine Size:  448 bytes,    Routine Base:  DBG$CODE + 0276
; 1138        1261  1 ROUTINE COPY_DESC_HANDLER (SIG, MECH) =
; 1139        1262  1 !
; 1140        1263  1 ! FUNCTION
; 1141        1264  1 !       This is the error hander for DBG$NCOPY_DESC. This routine is
; 1142        1265  1 !       responsible for freeing up the memory we have allocated so
; 1143        1266  1 !       far, if we get a NOFREE error message while copying the descriptor.
; 1144        1267  1 !
; 1145        1268  1 ! INPUTS
; 1146        1269  1 !       SIG     - Signal argument vector
; 1147        1270  1 !       MECH    - not used
; 1148        1271  1 !
; 1149        1272  1 ! IMPLICIT INPUT
; 1150        1273  1 !       COPY_DESC_HEAD - An own variable that points to the head of
; 1151        1274  1 !                        the descriptor copy.
; 1152        1275  1 !
; 1153        1276  1 ! OUTPUTS
; 1154        1277  1 !       This routine resignals the error.
; 1155        1278  1 !
; 1156        1279  2     BEGIN
; 1157        1280  2     MAP
; 1158        1281  2         SIG: REF VECTOR;
; 1159        1282  2
; 1160        1283  2     ! Only do something if the error is "no free storage" and if the own
; 1161        1284  2     ! variable COPY_DESC_HEAD is not zero (meaning that some storage has
; 1162        1285  2     ! been allocated before the NOFREE).
; 1163        1286  2     !
; 1164        1287  2     IF .SIG[1] EQL DBG$_NOFREE
; 1165        1288  2     THEN
; 1166        1289  2         IF .COPY_DESC_HEAD NEQ 0
; 1167        1290  2         THEN
; 1168        1291  3             BEGIN
; 1169        1292  3             DBG$NFREE_DESC(.COPY_DESC_HEAD);
; 1170        1293  3             COPY_DESC_HEAD = 0;
; 1171        1294  2             END;
; 1172        1295  2
; 1173        1296  2     ! Having freed the storage, resignal the error.
; 1174        1297  2     !
; 1175        1298  2     RETURN SS$_RESIGNAL;
; 1176        1299  1     END;



                                  0004 00000 COPY_DESC_HANDLER:
                                                   .WORD   Save R2                                   ; 1261
           52 00000000'     EF  9E 00002           MOVAB   COPY_DESC_HEAD, R2
           50          04   AC  D0 00009           MOVL    SIG, R0                                   ; 1287
   00028332  8F        04   A0  D1 0000D           CMPL    4(R0), #164658                            ;
                            0E  12 00015           BNEQ    1$                                        ;
           50               62  D0 00017           MOVL    COPY_DESC_HEAD, R0                        ; 1289
                            09  13 0001A           BEQL    1$                                        ;
                            50  DD 0001C           PUSHL   R0                                        ; 1292
      0000V  CF             01  FB 0001E           CALLS   #1, DBG$NFREE_DESC                        ;
                            62  D4 00023           CLRL    COPY_DESC_HEAD                            ; 1293
           50      0918     8F  3C 00025 1$:       MOVZWL  #2328, R0                                 ; 1298
                                04 0002A           RET                                               ; 1299
```

```
; Routine Size:  43 bytes,    Routine Base:  DBG$CODE + 0436
```

```
GLOBAL ROUTINE DBG$NFREE_DESC (DESC, PARAM2, PARAM3) =
!
!++
! FUNCTIONAL DESCRIPTION:
!
!       Releases dynamic storage associated with a non-volatile copy of a
!       language specific value or primary descriptor.
!       This routine accepts as input a copy of a primary or value
!       descriptor produced by DBG$NCOPY_DESC and calls the
!       routine DBG$REL_MEMORY to release each block of non-listed dynamic
!       storage contained within the descriptor.
!
!       This routine calls a language-specific routine based on the
!       language code in the descriptor header.
!
! FORMAL PARAMETERS:
!
!       desc                    - The address of a non-volatile primary or
!                                 value descriptor
!
!       param2                  - The address of a longword to contain the address
!                                 of a message argument vector for errors
!
! IMPLICIT INPUTS:
!
!       NONE
!
! IMPLICIT OUTPUTS:
!
!       On failure, a message argument vector.
!
! ROUTINE VALUE:
!
!       An unsigned integer longword completion code
!
! COMPLETION CODES:
!
!       STS$K_SUCCESS   (1)     - Success. Storage for  descriptor released.
!
!       STS$K_SEVERE    (4)     - Failure. Storage for descriptor not released. Message
!                                 argument vector constructed and returned.
!
! SIDE EFFECTS:
!
!       Dynamic memory is returned to the free storage pool.
!
!--
    BEGIN
    MAP
        DESC: REF DBG$VALDESC;

    ! Handle value descriptors separately from primary descriptors.
    !
    SELECTONE .DESC [DBG$B_DHDR_TYPE] OF
        SET

        ! Ordinary value descriptors. These are allocated in one contiguous
```

```
2          ! block so we can just release that block.
2          !
2          [DBG$K_VALUE_DESC, DBG$K_V_VALUE_DESC]:
3              BEGIN
3              DBG$REL_MEMORY (.DESC);
2              END;
2
2          ! New style Primary Descriptors. Here we have to release storage
2          ! for the root node and all the subnodes.
2          !
2          [DBG$K_PRIMARY_DESC]:
3              BEGIN
3              MAP
3                  DESC: REF DBG$PRIMARY;                          ! Pointer to the Primary
3                                                                  !     Descriptor for which
3                                                                  !     a symid list is to
3                                                                  !     be constructed.
3              LOCAL
3                  NEW_SUBNODE,                                    ! Pointer to the next
3                                                                  !     subnode
3                  SAVED_PTR,                                      ! Position in root node
3                                                                  !     that is pointed to
3                                                                  !     by the flink in the
3                                                                  !     last subnode.
3                  SUBNODE: REF DBG$PRIM_NODE;                     ! Pointer to a subnode.
3
3              ! First save away a pointer to the subnode and a pointer
3              ! which will identify when we have looped through all the
3              ! subnodes. Then release the storage associated with the
3              ! root node.
3              !
3              SAVED_PTR = DESC [DBG$L_PRIM_FLINK];
3              SUBNODE = .DESC [DBG$L_PRIM_FLINK];
3              DBG$REL_MEMORY (.DESC);
3
3              ! Loop through the subnodes. After saving a pointer to the
3              ! next subnode, release the storage for the current subnode.
3              !
3              WHILE .SUBNODE NEQ .SAVED_PTR DO
4                  BEGIN
4                  NEW_SUBNODE = .SUBNODE [DBG$L_PNODE_FLINK];
4                  DBG$REL_MEMORY (.SUBNODE);
4                  SUBNODE = .NEW_SUBNODE;
3                  END;
2              END;
2
2          ! At implementation level 3, we do not expect any other kind
2          ! of descriptor.
2          !
2          [OTHERWISE]:
2              $DBG_ERROR ('DBGLANVEC\DBG$NFREE_DESC');
2
2          TES;
2
2      ! The storage has been freed. Return success.
2      !
2      RETURN STS$K_SUCCESS;
```

```
; 1292           1414  1      END;


                                                                         .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

24  47  42  44  5C  43  45  56  4E  41  4C  47  42  44  18  0011E P.AAI:  .ASCII  <24>\DBGLANVEC\<92>\DBG$NFREE_DESC\          ;
                    43  53  45  44  5F  45  45  52  46  4E  0012D                                                                 ;


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

                                                    003C 00000          .ENTRY  DBG$NFREE_DESC, Save R2,R3,R4,R5        ; 1300
                        55 00000000G  00   9E 00002          MOVAB   DBG$REL_MEMORY, R5                              ;
                        52          04    AC   D0 00009          MOVL    DESC, R2                                        ; 1353
                        50          02    A2   9A 0000D          MOVZBL  2(R2), R0                                       ;
               7A       8F                50   91 00011          CMPB    R0, #122                                        ; 1359
                                          06   13 00015          BEQL    1$                                              ;
               83       8F                50   91 00017          CMPB    R0, #131                                        ;
                                          07   12 0001B          BNEQ    2$                                              ;
                                          52   DD 0001D 1$:      PUSHL   R2                                              ; 1361
                        65                01   FB 0001F          CALLS   #1, DBG$REL_MEMORY                              ;
                                          3A   11 00022          BRB     5$                                              ; 1353
               79       8F                50   91 00024 2$:      CMPB    R0, #121                                        ; 1367
                                          1F   12 00028          BNEQ    4$                                              ;
                        54          14    A2   9E 0002A          MOVAB   20(R2), SAVED_PTR                               ; 1388
                        53          14    A2   D0 0002E          MOVL    20(R2), SUBNODE                                 ; 1389
                                          52   DD 00032          PUSHL   R2                                              ; 1390
                        65                01   FB 00034          CALLS   #1, DBG$REL_MEMORY                              ;
                        54                53   D1 00037 3$:      CMPL    SUBNODE, SAVED_PTR                              ; 1395
                                          22   13 0003A          BEQL    5$                                              ;
                        52                63   D0 0003C          MOVL    (SUBNODE), NEW_SUBNODE                          ; 1397
                                          53   DD 0003F          PUSHL   SUBNODE                                         ; 1398
                        65                01   FB 00041          CALLS   #1, DBG$REL_MEMORY                              ;
                        53                52   D0 00044          MOVL    NEW_SUBNODE, SUBNODE                            ; 1399
                                          EE   11 00047          BRB     3$                                              ; 1395
                            00000000'     EF   9F 00049 4$:      PUSHAB  P.AAI                                           ; 1407
                                          01   DD 0004F          PUSHL   #1                                              ;
                            00028362      8F   DD 00051          PUSHL   #164706                                         ;
               00000000G  00              03   FB 00057          CALLS   #3, LIB$SIGNAL                                  ;
                        50                01   D0 0005E 5$:      MOVL    #1, R0                                          ; 1413
                                          04      00061          RET                                                     ; 1414

; Routine Size:  98 bytes,    Routine Base:  DBG$CODE + 0461
```

```
1 GLOBAL ROUTINE DBG$NGET_SYMID (DESC, PARAM2, PARAM3) =
1 !
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       Returns a list of symids contained within a language specific primary
1 !       or value descriptor.
1 !
1 !       This routine calls a language-specific routine based on the language
1 !       code in the descriptor header.
1 !
1 ! FORMAL PARAMETERS:
1 !
1 !       desc            - A longword containing the address of a language specific
1 !                         primary or value descriptor.
1 !
1 !       param2          - The address of a longword to contain the address of
1 !                         the first node in the symid list. Each node in the
1 !                         consists of a two longword block. The first longword
1 !                         is the link field and contains the address of the
1 !                         next node in the list. This field is 0 for the last
1 !                         node in the list. The second longword contains the
1 !                         value of a symid. Each symid that appears in a
1 !                         descriptor should appear once and only once in the
1 !                         symid list.
1 !
1 !       param3          - The address of a longword to contain the address of
1 !                         a message argument vector as described on page 4-119
1 !                         of the VAX/VMS system reference, volume 1A
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       NONE
1 !
1 ! IMPLICIT OUTPUTS:
1 !
1 !       In case of a severe error return, a message argument vector is constructed
1 !       from dynamic storage and returned.
1 !
1 ! ROUTINE VALUE:
1 !
1 !       An unsigned integer longword completion code
1 !
1 ! COMPLETION CODES:
1 !
1 !       STS$K_SUCCESS (1) - Success. Symid list constructed and returned.
1 !
1 !       STS$K_SEVERE  (4) - Failure. No symid list returned. Message argument
1 !                           vector constructed and returned.
1 !
1 ! SIDE EFFECTS:
1 !
1 !       NONE
1 !
2     BEGIN
2     MAP
2       DESC: REF DBG$VALDESC;
2     LOCAL
```

```
2               DIMCNT,
2               NUMBLKS,
2               SUBCNT,
2               SYMID_LIST;                             ! Pointer to the head of
2                                                       !    the symid list.
2
2       ! Implementation level 3 - all languages at this level have common
2       ! descriptors so we construct the symid list here.
2       !
2
2           ROUTINE APPEND_TO_LIST (SYMID, SYMID_LIST) : NOVALUE =
2           !
2           ! FUNCTION
2           !       This subroutine is used below to append a new symid
2           !       to the symid list under construction.
2           !
2           ! INPUTS
2           !       SYMID -         The symid to be added to the list.
2           !       SYMID_LIST -    Points to a longword containing a pointer
2           !                       to the head of the symid list.
2           !
2           ! OUTPUTS
2           !       If the symid list was empty, a one-node list will be
2           !       created and the SYMID_LIST parameter will contain
2           !       a pointer to this one-node list.
2           !       Otherwise, the SYMID_LIST parameter is left unchanged
2           !       but a node may be added to the list it points to.
2           !
3               BEGIN
3               LOCAL
3                   LINK_NODE: REF DBG$LINK_NODE,     ! Pointer to a node in the
3                                                     !    symid list.
3                   PREV_NODE: REF DBG$LINK_NODE;     ! Pointer to a node in the
3                                                     !    symid list.
3
3               ! If the symid is zero, do not add it to the list.
3               !
3               IF .SYMID EQL 0 THEN RETURN;
3
3               ! First check whether the given symid is on the list already.
3               !
3               LINK_NODE = ..SYMID_LIST;
3               PREV_NODE = .SYMID_LIST;
3               WHILE .LINK_NODE NEQ 0 DO
4                   BEGIN
4                   IF .LINK_NODE [DBG$L_LINK_NODE_VALUE] EQL .SYMID
4                   THEN
4                       RETURN;
4                   PREV_NODE = .LINK_NODE;
4                   LINK_NODE = .LINK_NODE [DBG$L_LINK_NODE_LINK];
3                   END;
3
3
3               ! Allocate space for a new node and put it on the list.
3               !
3               LINK_NODE = DBG$GET_TEMPMEM (DBG$K_LINK_NODE_SIZE);
3               PREV_NODE [DBG$L_LINK_NODE_LINK] = .LINK_NODE;
```

```
; 1408          1529  3              LINK_NODE [DBG$L_LINK_NODE_VALUE] = .SYMID;
; 1409          1530  2              END;



                                          000C 00000 APPEND_TO_LIST:
                                                              .WORD   Save R2,R3                                       ; 1482
                        52        04   AC  D0 00002           MOVL    SYMID, R2                                        ; 1509
                                       2A  13 00006           BEQL    3$                                               ;
                        50        08   BC  D0 00008           MOVL    @SYMID_LIST, LINK_NODE                           ; 1513
                        53        08   AC  D0 0000C           MOVL    SYMID_LIST, PREV_NODE                            ; 1514
                                       50  D5 00010 1$:       TSTL    LINK_NODE                                        ; 1515
                                       0E  13 00012           BEQL    2$                                               ;
                        52        04   A0  D1 00014           CMPL    4(LINK_NODE), R2                                 ; 1517
                                       18  13 00018           BEQL    3$                                               ;
                        53             50  D0 0001A           MOVL    LINK_NODE, PREV_NODE                             ; 1520
                        50             60  D0 0001D           MOVL    (LINK_NODE), LINK_NODE                           ; 1521
                                       EE  11 00020           BRB     1$                                               ; 1515
                                       02  DD 00022 2$:       PUSHL   #2                                               ; 1527
                00000000G 00           01  FB 00024           CALLS   #1, DBG$GET_TEMPMEM                              ;
                        63             50  D0 0002B           MOVL    LINK_NODE, (PREV_NODE)                           ; 1528
                  04    A0             52  D0 0002E           MOVL    R2, 4(LINK_NODE)                                 ; 1529
                                       04  00032 3$:          RET                                                      ; 1530

; Routine Size:  51 bytes,    Routine Base:  DBG$CODE + 04C3

; 1410          1531  2
; 1411          1532  2      ! Initialize the pointer to the symid list.
; 1412          1533  2      !
; 1413          1534  2      SYMID_LIST = 0;
; 1414          1535  2
; 1415          1536  2      ! Handle value descriptors separately from primary descriptors.
; 1416          1537  2      !
; 1417          1538  2      SELECTONE .DESC [DBG$B_DHDR_TYPE] OF
; 1418          1539  2          SET
; 1419          1540  2
; 1420          1541  2          ! Ordinary value descriptors.
; 1421          1542  2          !
; 1422          1543  2          [DBG$K_VALUE_DESC, DBG$K_V_VALUE_DESC]:
; 1423          1544  3              BEGIN
; 1424          1545  3              MAP
; 1425          1546  3                  DESC: REF DBG$VALDESC;  ! Pointer to a new style value
; 1426          1547  3                                          !     descriptor
; 1427          1548  3              APPEND_TO_LIST (.DESC [DBG$L_DHDR_TYPEID], SYMID_LIST);
; 1428          1549  3              APPEND_TO_LIST (.DESC [DBG$L_DHDR_SYMID0], SYMID_LIST);
; 1429          1550  2              END;
; 1430          1551  2
; 1431          1552  2          ! New style Primary Descriptors. Here we have to get symids from
; 1432          1553  2          ! the root node and all sub-nodes.
; 1433          1554  2          !
; 1434          1555  2          [DBG$K_PRIMARY_DESC]:
; 1435          1556  3              BEGIN
; 1436          1557  3
; 1437          1558  3              MAP
; 1438      1559  3          DESC: REF DBG$PRIMARY;                        ! Pointer to the Primary
; 1439      1560  3                                                        !      Descriptor for which
; 1440      1561  3                                                        !      a symid list is to
; 1441      1562  3                                                        !      be constructed.
; 1442      1563  3
; 1443      1564  3      LOCAL
; 1444      1565  3          SUBNODE: REF DBG$PRIM_NODE;                   ! Pointer to a subnode.
; 1445      1566  3
; 1446      1567  3      ! Append the typeid and the symid from the root node.
; 1447      1568  3      !
; 1448      1569  3      APPEND_TO_LIST (.DESC [DBG$L_DHDR_TYPEID], SYMID_LIST);
; 1449      1570  3      APPEND_TO_LIST (.DESC [DBG$L_DHDR_SYMID0], SYMID_LIST);
; 1450      1571  3
; 1451      1572  3      ! Loop through each of the subnodes.
; 1452      1573  3      !
; 1453      1574  3      SUBNODE = .DESC [DBG$L_PRIM_FLINK];
; 1454      1575  3      WHILE .SUBNODE NEQ DESC[DBG$L_PRIM_FLINK] DO
; 1455      1576  4          BEGIN
; 1456      1577  4
; 1457      1578  4          ! All kinds of subnodes have typeids and symids
; 1458      1579  4          ! so we append these.
; 1459      1580  4          !
; 1460      1581  4          APPEND_TO_LIST (.SUBNODE [DBG$L_PNODE_TYPEID], SYMID_LIST);
; 1461      1582  4          APPEND_TO_LIST (.SUBNODE [DBG$L_PNODE_SYMID], SYMID_LIST);
; 1462      1583  4
; 1463      1584  4          ! If the subnode is an array node then it also
; 1464      1585  4          ! has typeids in the subscript vector.
; 1465      1586  4          !
; 1466      1587  4          IF .SUBNODE [DBG$B_PNODE_FCODE] EQL RST$K_TYPE_ARRAY
; 1467      1588  4          THEN
; 1468      1589  5              BEGIN
; 1469      1590  5              LOCAL
; 1470      1591  5                  SUBVECTOR: REF DBG$PRIM_NODE_SUBS;
; 1471      1592  5              APPEND_TO_LIST (.SUBNODE [DBG$L_PNARR_CELLTYPE],SYMID_LIST);
; 1472      1593  5              SUBVECTOR = SUBNODE [DBG$A_PNARR_SVECTOR];
; 1473      1594  5              ! Use whichever is larger, subcnt or dimcnt.
; 1474      1595  5              SUBCNT = .SUBNODE[DBG$B_PNARR_SUBCNT];
; 1475      1596  5              DIMCNT = .SUBNODE[DBG$B_PNARR_DIMCNT];
; 1476      1597  5              IF .SUBCNT GTR .DIMCNT
; 1477      1598  5              THEN
; 1478      1599  5                  NUMBLKS = .SUBCNT
; 1479      1600  5              ELSE
; 1480      1601  5                  NUMBLKS = .DIMCNT;
; 1481      1602  5              INCR I FROM 0 TO .NUMBLKS-1 DO
; 1482      1603  5                  APPEND_TO_LIST (.SUBVECTOR[.I, DBG$L_PNSUB_TYPEID],
; 1483      1604  5                                  SYMID_LIST);
; 1484      1605  5              END
; 1485      1606  4          ELSE IF .SUBNODE [DBG$B_PNODE_FCODE] EQL RST$K_TYPE_VARIANT
; 1486      1607  4          THEN
; 1487      1608  4              APPEND_TO_LIST(.SUBNODE[DBG$L_PNVAR_TAGID],SYMID_LIST);
; 1488      1609  4
; 1489      1610  4          SUBNODE = .SUBNODE [DBG$L_PNODE_FLINK];
; 1490      1611  3          END;
; 1491      1612  2      END;
; 1492      1613  2
; 1493      1614  2  ! At implementation level 3, we do not expect any other kind
; 1494      1615  2  ! of descriptor.
; 1495          1616  2             !
; 1496          1617  2             [OTHERWISE]:
; 1497          1618  2                 $DBG_ERROR ('DBGLANVEC\DBG$NGET_SYMID');
; 1498          1619  2
; 1499          1620  2             TES;
; 1500          1621  2
; 1501          1622  2         ! The symid list has been constructed. Return success.
; 1502          1623  2         !
; 1503          1624  2         .PARAM2 = .SYMID_LIST;
; 1504          1625  2         RETURN STS$K_SUCCESS;
; 1505          1626  1         END;



                                                                          .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

24  47  42  44  5C  43  45  56  4E  41  4C  47  42  44  18  00137 P.AAJ:  .ASCII  <24>\DBGLANVEC\<92>\DBG$NGET_SYMID\                 ;
                    44  49  4D  59  53  5F  54  45  47  4E  00146                                                                  ;



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

                                                     03FC  00000          .ENTRY  DBG$NGET_SYMID, Save R2,R3,R4,R5,R6,R7,R8,-  ; 1415
                                                                                  R9                                           ;
                                    59        C8   AF   9E  00002         MOVAB   APPEND_TO_LIST, R9                           ;
                                                   7E   D4  00006         CLRL    SYMID_LIST                                   ; 1534
                                    55        04   AC   D0  00008         MOVL    DESC, R5                                     ; 1538
                               7A   8F        02   A5   91  0000C         CMPB    2(R5), #122                                  ; 1543
                                                   07   13  00011         BEQL    1$                                           ;
                               83   8F        02   A5   91  00013         CMPB    2(R5), #131                                  ;
                                                   13   12  00018         BNEQ    2$                                           ;
                                                   5E   DD  0001A  1$:    PUSHL   SP                                           ; 1548
                                              08   A5   DD  0001C         PUSHL   8(R5)                                        ;
                                    69             02   FB  0001F         CALLS   #2, APPEND_TO_LIST                           ;
                                                   5E   DD  00022         PUSHL   SP                                           ; 1549
                                              0C   A5   DD  00024         PUSHL   12(R5)                                       ;
                                    69             02   FB  00027         CALLS   #2, APPEND_TO_LIST                           ;
                                                 00A0   31  0002A         BRW     12$                                          ; 1538
                               79   8F        02   A5   91  0002D  2$:    CMPB    2(R5), #121                                  ; 1555
                                                   03   13  00032         BEQL    3$                                           ;
                                                 0081   31  00034         BRW     11$                                          ;
                                                   5E   DD  00037  3$:    PUSHL   SP                                           ; 1569
                                              08   A5   DD  00039         PUSHL   8(R5)                                        ;
                                    69             02   FB  0003C         CALLS   #2, APPEND_TO_LIST                           ;
                                                   5E   DD  0003F         PUSHL   SP                                           ; 1570
                                              0C   A5   DD  00041         PUSHL   12(R5)                                       ;
                                    69             02   FB  00044         CALLS   #2, APPEND_TO_LIST                           ;
                                    52        14   A5   D0  00047         MOVL    20(R5), SUBNODE                              ; 1574
                                    50        14   A5   9E  0004B  4$:    MOVAB   20(R5), R0                                   ; 1575
                                    50             52   D1  0004F         CMPL    SUBNODE, R0                                  ;
                                                   79   13  00052         BEQL    12$                                          ;
                                                   5E   DD  00054         PUSHL   SP                                           ; 1581
                                              0C   A2   DD  00056         PUSHL   12(SUBNODE)                                  ;
                                    69             02   FB  00059         CALLS   #2, APPEND_TO_LIST                           ;
                                                   5E   DD  0005C         PUSHL   SP                                           ; 1582
                                              10   A2   DD  0005E         PUSHL   16(SUBNODE)                                  ;
```

```
                         69              02  FB 00061              CALLS   #2, APPEND_TO_LIST                       ;
                         01        09    A2  91 00064              CMPB    9(SUBNODE), #T                           ; 1587
                                         3B  12 00068              BNEQ    9$                                       ;
                                         5E  DD 0006A              PUSHL   SP                                       ; 1592
                                   24    A2  DD 0006C              PUSHL   36(SUBNODE)                              ;
                         69              02  FB 0006F              CALLS   #2, APPEND_TO_LIST                       ;
                         53        28    A2  9E 00072              MOVAB   40(R2), SUBVECTOR                        ; 1593
                         56        1F    A2  9A 00076              MOVZBL  31(SUBNODE), SUBCNT                      ; 1595
                         58        1B    A2  9A 0007A              MOVZBL  27(SUBNODE), DIMCNT                      ; 1596
                         58              56  D1 0007E              CMPL    SUBCNT, DIMCNT                           ; 1597
                                         05  15 00081              BLEQ    5$                                       ;
                         57              56  D0 00083              MOVL    SUBCNT, NUMBLKS                          ; 1599
                                         03  11 00086              BRB     6$                                       ;
                         57              58  D0 00088 5$:          MOVL    DIMCNT, NUMBLKS                          ; 1601
                         54              01  CE 0008B 6$:          MNEGL   #1, I                                    ; 1603
                                         0F  11 0008E              BRB     8$                                       ;
                                         5E  DD 00090 7$:          PUSHL   SP                                       ;
            50           54              14  C5 00092              MULL3   #20, I, R0                               ;
                                   10 A043   9F 00096              PUSHAB  16(R0)[SUBVECTOR]                        ;
                                         9E  DD 0009A              PUSHL   @(SP)+                                   ;
                         69              02  FB 0009C              CALLS   #2, APPEND_TO_LIST                       ;
            ED           54              57  F2 0009F 8$:          AOBLSS  NUMBLKS, I, 7$                           ;
                                         0E  11 000A3              BRB     10$                                      ; 1587
                         13        09    A2  91 000A5 9$:          CMPB    9(SUBNODE), #19                          ; 1606
                                         08  12 000A9              BNEQ    10$                                      ;
                                         5E  DD 000AB              PUSHL   SP                                       ; 1608
                                   1C    A2  DD 000AD              PUSHL   28(SUBNODE)                              ;
                         69              02  FB 000B0              CALLS   #2, APPEND_TO_LIST                       ;
                         52              62  D0 000B3 10$:         MOVL    (SUBNODE), SUBNODE                       ; 1610
                                         93  11 000B6              BRB     4$                                       ; 1575
                              00000000'  EF  9F 000B8 11$:         PUSHAB  P.AAJ                                    ; 1618
                                         01  DD 000BE              PUSHL   #1                                       ;
                              00028362   8F  DD 000C0              PUSHL   #164706                                  ;
               00000000G  00             03  FB 000C6              CALLS   #3, LIB$SIGNAL                           ;
                      08  BC             6E  D0 000CD 12$:         MOVL    SYMID_LIST, @PARAM2                      ; 1624
                          50             01  D0 000D1              MOVL    #1, R0                                   ; 1625
                                             04 000D4              RET                                              ; 1626

; Routine Size:  213 bytes,    Routine Base:  DBG$CODE + 04F6
; 1507          1627  1 GLOBAL ROUTINE DBG$NINITIALIZE : NOVALUE =
; 1508          1628  1 !
; 1509          1629  1 ! FUNCTION
; 1510          1630  1 !       This routine calls language specific initialization routines.  This is
; 1511          1631  1 !       done before each command is processed to garuantee the integrity of the
; 1512          1632  1 !       language specific machinery.
; 1513          1633  1 !
; 1514          1634  1 ! FORMAL PARAMETERS:
; 1515          1635  1 !
; 1516          1636  1 !       NONE
; 1517          1637  1 !
; 1518          1638  1 ! IMPLICIT INPUTS:
; 1519          1639  1 !
; 1520          1640  1 !       NONE
; 1521          1641  1 !
; 1522          1642  1 ! IMPLICIT OUTPUTS:
; 1523          1643  1 !
; 1524          1644  1 !       NONE
; 1525          1645  1 !
; 1526          1646  1 ! ROUTINE VALUE:
; 1527          1647  1 !
; 1528          1648  1 !       NOVALUE
; 1529          1649  1 !
; 1530          1650  1 ! COMPLETION CODES:
; 1531          1651  1 !
; 1532          1652  1 !       NONE
; 1533          1653  1 !
; 1534          1654  1 ! SIDE EFFECTS:
; 1535          1655  1 !
; 1536          1656  1 !       NONE
; 1537          1657  1 !
; 1538          1658  2   BEGIN
; 1539          1659  2   RETURN;
; 1540          1660  1   END;



                                                  0000 00000        .ENTRY  DBG$NINITIALIZE, Save nothing                  ; 1627
                                                    04 00002        RET                                                    ; 1660

; Routine Size:  3 bytes,    Routine Base:  DBG$CODE + 05CB


; 1541          1661  0 END 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                        336  NOVEC,NOWRT,  RD ,  EXE,  SHR,  LCL,  REL,  CON,  PIC,ALIGN(0)
;  DBG$CODE                       1486  NOVEC,NOWRT,  RD ,  EXE,  SHR,  LCL,  REL,  CON,  PIC,ALIGN(0)


;                            Library Statistics
;
;                                                --------- Symbols --------          Pages        Processing
;        File                                    Total    Loaded   Percent          Mapped        Time
;
;   _$255$DUA28:[SYSLIB]LIB.L32;1                18619         6         0           1000         00:01.9
;   _$255$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1           32         0         0              7         00:00.1
;   _$255$DUA28:[DEBUG.OBJ]DBGLIB.L32;1           1545       110         7             97         00:02.0
;   _$255$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1
;                                                  418         3         0             31         00:00.3
;   _$255$DUA28:[DEBUG.OBJ]DBGMSG.L32;1            386         2         0             22         00:00.3




;                                  COMMAND QUALIFIERS

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

; Size:           1486 code + 340 data bytes
; Run Time:          00:32.0
; Elapsed Time:      01:49.9
; Lines/CPU Min:      3116
; Lexemes/CPU-Min:  11001
; Memory Used:  181 pages
; Compilation Complete
```

0084 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

DBGIFTHEN
LIS

DBGLANVEC
LIS

DBGLANGOP
LIS

DBGGEN
LIS

DBGLEVEL1
LIS